VISIT…

Design and Stop Exercises Solutions

to accompany the third edition of

C++ for Computer Science and Engineering

by Vic Broquard

© 2001, 2002, 2003 by Vic Broquard, All rights reserved

Published by Broquard eBooks
103 Timberlane
East Peoria, IL 61611

Chapter 1 — Design Exercises

1. How would you solve this problem? What is the answer? A bug wishes to climb to the top of a 12-foot tall telephone pole. During the day, it climbs 3 feet. However, while it sleeps at night, the bug slides back down 2 feet. How many days does it take the bug to reach its objective, the top of the pole?

Answer:

day	Starts Day At	Height Made that Day
1	**0**	**3**
2	**1**	**4**
3	**2**	**5**
4	**3**	**6**
5	**4**	**7**
6	**5**	**8**
7	**6**	**9**
8	**7**	**10**
9	**8**	**11**
10	**9**	**12 — done he's at the top**

2. Sketch a solution in pseudocode or English to solve this problem. A math teacher wishes to have a program that displays the multiplication tables for her fourth graders. She wants the program to accept any whole number (integer) from 1 to 9. The program then displays the multiplication tables from 1 to that number. A sample run might be as follows. Note she enters the underlined number 4.

```
Enter a number from 1 to 9: 4
1 x 1 = 1 x 1 = 1
1 x 2 = 2 x 1 = 2
1 x 3 = 3 x 1 = 3
1 x 4 = 4 x 1 = 4
2 x 2 = 2 x 2 = 4
2 x 3 = 3 x 2 = 6
2 x 4 = 4 x 2 = 8
3 x 3 = 3 x 3 = 9
3 x 4 = 4 x 3 = 12
4 x 4 = 4 x 4 = 16
```

A Solution:

```
display "Enter a number from 1 to 9: "
input endnumber
let startnumber = 1
do the following while startnumber is less than or equal to endnumber
        let currentnumber = startnumber
```

```
            do the following while currentnumber is less than or equal to endnumber
                display startnumber, 'x', currentnumber, '=', currentnumber, 'x',
                        startnumber, '=' currentnumber times startnumber
                add 1 to currentnumber
            end do
            add one to startnumber
        end do

main storage: startnumber endnumber currentnumber
```

3. Sketch a solution in pseudocode or English to solve this problem. A manager of some carpet store wishes a program that calculates the square footage of carpet a customer requires and determines his cost for installation based on the square footage. The program first asks him to enter the length and width of the room. It then displays the square-footage. His installation cost is found by multiplying the square footage by 7.5%. A test run might be:

Enter the length and width of the carpet: 10 20
The square footage is 200 and the service charge is $15.00

solution:

```
display "Enter the length and width of the carpet: "
input length and width
sqFootage = length times width
serviceCharge = sqFootage times .075
display "The square footage is ", sqFootage,
        " and the service charge is $", serviceCharge

main storage: length   width   sqFootage   serviceCharge
```

Chapter 1 — Stop! Do These Exercises Before Programming

Correct the errors in the following programs. If you are having trouble determining what is wrong, you can always make a test program, enter this coding and see what the compiler indicates is wrong.

1. Why does this program not compile? Show what must be done to fix it?

```
int main () {
  cout << "Hi there!\n";
  return 0;
}
#include <iostream>
#include <iomanip>
```

solution: header includes must come first and needs the using namespace statement

```
#include <iostream>
#include <iomanip>
using namespace std;
int main () {
  cout << "Hi there!\n";
  return 0;
}
```

2. Why does this program not compile? Show what must be done to fix it?

```
#include <iostream>
#include <iomanip>
Int Main () {
  Cout << "Great day outside!!\n";
  return 0;
}
```

solution: you must watch the case of items and using namespace

```
#include <iostream>
#include <iomanip>
using namespace std;
int main () {
  cout << "Great day outside!!\n";
  return 0;
}
```

3. Why does this program not compile? Show what must be done to fix it?

```
#include <iostream>
#include <iomanip>
```

```
using namespace std;
int main () {
  cout << Hi there! << endl;
  return 0;
}
```

solution: strings are enclosed in “ ”

```
#include <iostream>
#include <iomanip>
using namespace std;
int main () {
  cout << "Hi there!" << endl;
  return 0;
}
```

4. Why does this program not produce any output? Show what must be done to fix it.

```
#include <iostream>
#include <iomanip>
Using Namespace Std;
int main () {
  return 0;
  cout << "John Jones successfully made this" << endl;
}
```

solution: return ends the function and it never gets to the **cout** - watch case on using statement

```
#include <iostream>
#include <iomanip>
using namespace std;
int main () {
  cout << "John Jones successfully made this" << endl;
  return 0;
}
```

5. Why does this program not compile? Show what must be done to fix it?

```
#include <iostream>
#include <iomanip>
using namespace std
int main ()
  cout << "Something is very wrong here" << endl;
  return 0;
}
```

solution: a block of coding begins with a { and ; after using statement

```
#include <iostream>
#include <iomanip>
using namespace std;
```

```
int main () {
  cout << "Something is very wrong here" << endl;
  return 0;
}
```

6. Why does this program not compile? Show what must be done to fix it?

```
#include <iostream>
#include <iomanip>
using namespace std;
int main (){
  c out >> "Something is still wrong here" << endl;
  Return zero;
}
```

solution: watch out for case and blanks which separate things

```
#include <iostream>
#include <iomanip>
using namespace std;
int main (){
  cout >> "Something is still wrong here" << endl;
  return 0;
}
```

7. Why does this program not compile? Show what must be done to fix it?

```
#include <iostream>
#include <manip>
using namespace std;
int main (){
  cout << 'I cannot get this program to work!' << << endl;
  return 0;
}
```

solution: watch the spelling of the include filenames; strings are surrounded by “” not ‘ ’

```
#include <iostream>
#include <iomanip>
using namespace std;
int main (){
  cout << "I cannot get this program to work!" << << endl;
  return 0;
}
```

8. This program compiles, but what is stylistically wrong? Show what must be done to fix it?

```
#include <iostream>
#include <iomanip>
using namespace std;
int main () {
cout << "Something is not quite right here" << endl;
return 0;}
```

solution: no indentation and incorrect alignment of } and could use blank lines for readability

```
#include <iostream>
#include <iomanip>
using namespace std;

int main () {
     cout << "Something is not quite right here" << endl;
     return 0;
}
```

9. A programmer has written the following solution to calculate wages for their company's weekly payroll. The programmer is convinced it will work just fine and has submitted it to you for verification and approval. This means you must thoroughly desk check their solution. How many errors can you find in it? Show how could they be repaired so that it would work.

Each line of input contains the employee's number, the hours they have worked and their pay rate. Any hours more than 40 are to be paid at time and a half.

Set **total_payroll** to 0
input **employee_id** and **hours** and **payrate**
as long as we got a set of data, do the following
    multiply **hours** by **payrate** and store it in **pay**
    if the **hours** is greater than 40 then do this
        Pay = (**hours** - 40) times **rate** times 1.5
    end the if
    add the **pay** to the **TotalPayroll**
    display the **id number** and the **pay**
    try to input another **employee_id** and **hours** and **payrate**
end the do series here
display the **total_payroll**

Test the program with the following input lines of data
123455 40 5.00
245346 20 7.50
535323 60 6.00

Hint: draw a picture of what main storage or memory should be. Pay attention to the names of the variables used in the solution.

Solution: test it first
main memory:

	total_payroll	employee_id	hours	payrate	pay	Pay	TotalPayroll	id number
test 1	0	123455	40	5.00	200.00	??	???+200	????

oops we have used multiple names for the same thing - so change names to all agree

	total_payroll	employee_id	hours	payrate	pay
test 1	200.00	123455	40	5.00	200.00

Displays 123455 200.00
Version 1 - changes in bold - produces ok output for test 1

Set total_payroll to 0
input employee_id and hours and payrate
as long as we got a set of data, do the following
multiply hours by payrate and store it in pay
if the hours is greater than 40 then do this
pay = (hours - 40) times **payrate** times 1.5
end the if
add the pay to the total_payroll
display the employee_id and the pay
try to input another employee_id and hours and payrate
end the do series here
display the total_payroll

Now trying test 2

	total_payroll	employee_id	hours	payrate	pay
test 2	150.0 0	245346	20	7.50	150.00

Displays: 245346 150.00
Tests ok

Now trying test 3

	total_payroll	employee_id	hours	payrate	pay
test 2	108.0 0	535323	60	6.00	108.00

Displays: 245346 108.00
Oops - short changed their pay!
Version 2 - changes in bold - produces ok output for test 1

Set total_payroll to 0
input employee_id and hours and payrate
as long as we got a set of data, do the following
if the hours is greater than 40 then do this
pay = 40 times payrate + (hours - 40) times payrate times 1.5
otherwise do this

multiply hours by payrate and store it in pay
end the if
add the pay to the total_payroll
display the employee_id and the pay
try to input another employee_id and hours and payrate
end the do series here
display the total_payroll

Now trying test 3 again

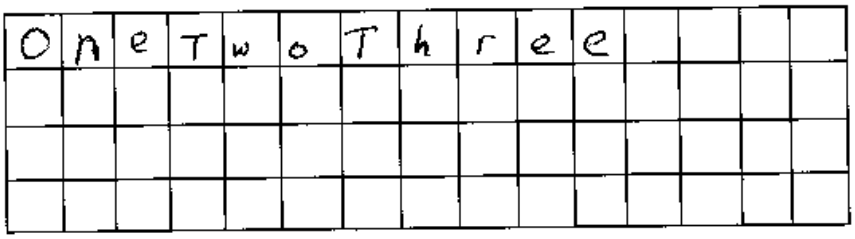
	total_payroll	employee_id	hours	payrate	pay
test 2	420.00	535323	60	6.00	420.00

Displays: 245346 420.00

What is the precise output from these output instructions? Fill in the boxes to indicate the results. Each box represents one column on the screen and each row represents one line on the screen.

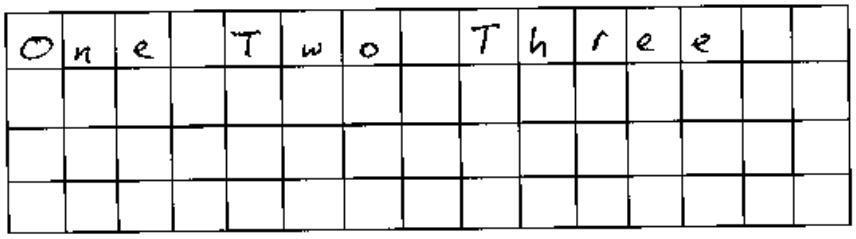
10.

```
cout << "One";
cout << "Two";
cout << "Three";
```



11.

```
cout << "One ";
cout << "Two ";
cout << "Three";
```

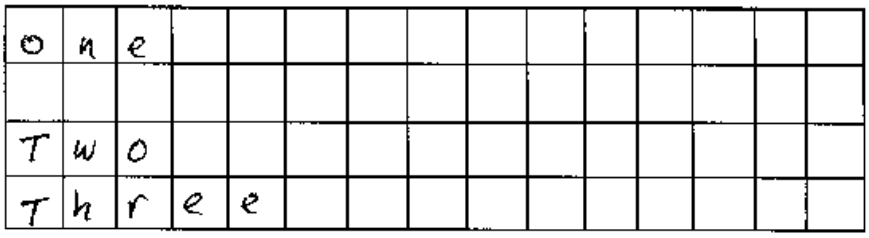


12.

```
cout << "One" << endl;
cout << "Two" << endl;
cout << "Three" << endl;
```

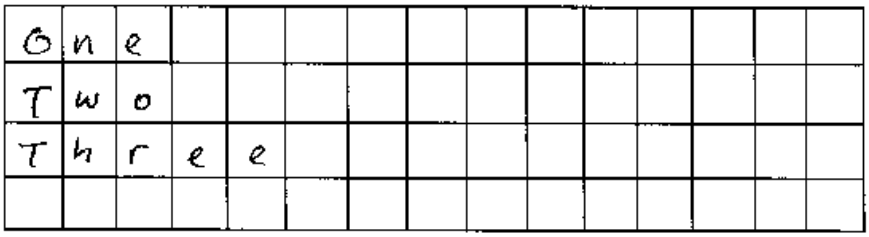
13.

```
cout << "One\n";
cout << "\nTwo\n";
cout << "Three";
```



14.

```
cout << "One\nTwo\nThree\n";
```



Chapter 2 — Design Exercises

1. Mysterious “crop circles” sometimes appear in a farmer’s corn field. A crop circle is an area in the middle of his corn field in which all of the corn stalks have been trampled flat, yielding some design visible only from the air. Farmer Jones discovered just such a circle in his field. Since the plants were smashed, Farmer Jones suffers a crop yield loss. His crop insurance covers some of his lost income by paying him a rate of $2.25 per bushel of corn lost. His yield on the remainder of that field is 125 bushels per acre. He measured the crop circle and found it was 50 feet in diameter. How much money does he get from the crop insurance? Hint, the area of a circle is given by PI times the square of the radius and an acre is 4840 square yards.

A Solution:

let PI = 3.14159
let radius = 50. / 2.
let area = PI * radius * radius
let areaSqYards = area / 9
let areaAcres = areaSqYards / 4840
let yieldLoss = areaAcre * 125
let lostIncome = yieldLoss * 2.25
$12.68

2. It’s party time. You are planning a party and have $40 with which to buy as many refreshments as possible. But not every guest prefers the same refreshments. Six guests prefer pizza while eight prefer to eat a hot dog. Four guests like Pepsi, eight prefer Coca-Cola, and two want Dr. Pepper. A pizza comes with eight large slices and costs $9.00. Hot dogs cost $1.25 each. A six-pack of any kind of soda pop costs $3.50. The rules you must follow are:

All guests must have something they like to eat and drink.
Soda pop can only be purchased in six-packs.
Pizza can only be bought as a whole pizza with eight slices.

What is the minimum that you must buy to satisfy these criteria? Do you have enough money to pay for it? (Ignore sales taxes.)

A solution:

Apply rule 1.
6 pizza, 8 hotdogs, 4 Pepsi, 8 Cokes, 2 Dr. Peppers
Apply rule 2 and 3 to the above results
1 large pizza, 8 hotdogs, 1 6-pack Pepsi, 2 6-pack Coke, 1 6-pack Dr.Pepr
Yields: 9.00 + 8 * 1.25 + 4 * 3.50 = $33.00 and you have 7.00 left over Party-On!

Chapter 2 - Stop! Do These Exercises Before Programming

1. The programmer was having a bad day writing this program. Correct the errors so that there are no compile time errors. Make whatever assumptions you can about data types that need fixing up.

```
#includ <iostrea>
Const int TOTAL 100;
int main () {
 Double costOfItem;
 quantity = 42;
 double total cost;
 cupon_discount int;
 const double AmountPaid;
 cost of item = 4.99
 AmountPaid = 9.99;
 ...
```

Solution:

```
#include <iostream>
using namespace std;
const int TOTAL = 100;
int main () {
 double costOfItem;
 int quantity = 42;
 double totalCost;
 int coupon_discount;
 const double AmountPaid = 9.99;
 costOfItem = 4.99
 ...
```

2. Circle the variable names that are **invalid** C++ variable names. Do not circle ones that are legal, even though they might not represent the best naming convention.

```
CostOfGoodsSold   Ok
total Price       Illegal
C3P0              Ok
3D-Movie          Illegal
distance Traveled Illegal
sin               Ok but lousy conflicts with math function
fAbs              Ok but lousy
Log               Ok but lousy
qty_sold          Ok
qty sold          Illegal
qtySold           Ok
```

3. Convert each of these formulas into a proper C++ statement.

a. F = m a (Force = mass times acceleration)
F = m * a;

b. A = PI R^2 (area of a circle)
A = PI * R * R;

c. $$total = \frac{n(n+1)}{2}$$

total = n * (n + 1) / 2;

d. x = sin (2 PI y);
x = sin (2 * PI * y);

e. $$z = \sqrt{x^2 + y^2}$$

z = sqrt (x * x + y * y);

f. $$x = \frac{-B + \sqrt{B^2 - 4AC}}{2A}$$

x = (-B + sqrt (B * B - 4 * A * C)) / (2 * A);

g. $$x = \sqrt[4]{z}$$

x = pow (z, .25);

h. $$y = ae^x + b$$

y = a * exp (x) + b;

4. Correct the errors in these C++ calculations.

a. cost = qty unitPrice; // cost is qty times unitPrice
cost = qty * unitPrice;

b. sum = sum + + count; // add count to sum
sum = sum + count; or sum += count;

c. count + 1 = count // add one to count
count = count + 1; or count++; or count += 1;

d. root = sqrt x * x + y * y; // x is the square root of x squared + y squared
root = sqrt (x * x + y * y);

e. xy = Pow (x, y); // calculate x raised to the yth power
xy = pow (x, y);

f. count + 1; // increment count
count++; or **count += 1;**

5. The equation to be solved is this

$$percent1 = \frac{salesTotal1}{salesTotal1 + salesTotal2} 100$$

Assuming all variables are **double**s, which of the following correctly calculates the percentage? Next, assuming all variables are **integer**s, which of the following correctly calculates the percentage? Indicate which work for doubles and which work for integers by placing an I or a D before each letter.

a. percent1 = salesTotal1 / salesTotal1 + salesTotal2 * 100;
fails for both types

b. percent1 = salesTotal1 / (salesTotal1 + salesTotal2 * 100);
fails for both types

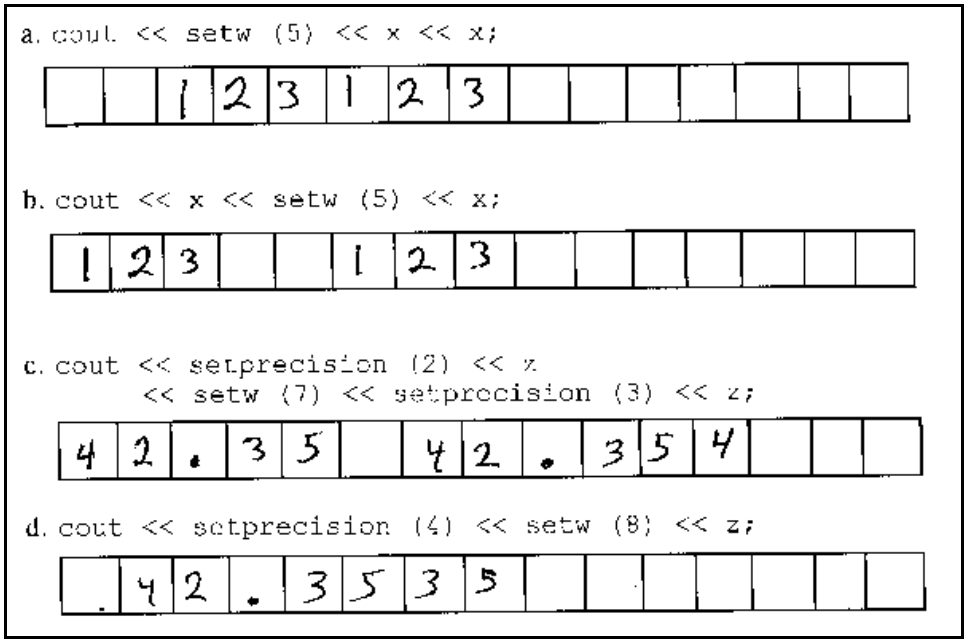
c. percent1 = salesTotal1 / (salesTotal1 + salesTotal2) * 100;
works for doubles; fails for ints

d. percent1 = ((salesTotal1) / (salesTotal1 + salesTotal2)) * 100;
works for doubles; fails for ints
e. percent1 = salesTotal1 * 100 / salesTotal1 + salesTotal2;
fails for both types

f. percent1 = salesTotal1 * 100 / (salesTotal1 + salesTotal2);
works for both types

6. Show the precise output from the following series of **cout** instructions. Assume these are the initial values of the variables. Assume the **ios::fixed** has been set along with **ios::showpoint**. The precision has not been set to any value initially.

```
int x = 123;
double z = 42.35353;
```



7. For each of these short calculations, show the result that is displayed. Assume that **ios::fixed** and **ios::showpoint** have been set on **cout** and that the precision is set to two decimal digits unless overridden.

a.

```
int x = 10;
int y = 4;
cout << x / y;
```

ansr: 2

b.

```
int pennies = 123;
const int QUARTERS = 25;
int quarters;
quarters = pennies / QUARTERS;
pennies = pennies % QUARTERS;
cout << quarters << " " << pennies;
```

ansr: 4 23

c.

```
double number = 100;
cout << sqrt (number);
```

ansr: 10.00

d.

```
double num = 10;
double bignum;
bignum = pow (num, 2);
cout << setprecision (0) << bignum;
```

ansr: 100.

Chapter 3 — Design Exercises

1. Sketch the pseudocode to solve this problem. The user enters some even integer greater than two, called say number. The program determines the next higher even integer from number and the next lower even integer from number. Display the sum of the three numbers, the product of the three numbers, and the average of the three numbers.

A solution:

Main storage:

```
     number   nextHiEven  nextLoEven   sum   product   avg
test    4         6          2         12     48       4
        input number
        nextHiEven = number + 2;
        nextLoEven = number - 2;
        sum = number + nextHiEven + nextLoEven;
        avg = sum / 3;
        product = number * nextHiEven + nextLoEven;
        display sum, product, avg
```

2. Sketch the pseudocode to solve this problem. The user wishes to enter six temperature observations taken at four-hour intervals throughout the day. Compute and print the average temperature for the day.

A solution:

main storage

```
        temp1   temp2   temp3   temp4   temp5   temp6
a test    10      12      14      16      18      20
displays 15

        display "enter six temperatures separated by a blank\n"
        input temp1, temp2, temp3, temp4, temp5, temp6
        sum = temp1 + temp2 + temp3 + temp4 + temp5 + temp6
        avgTemp = sum / 6
        display avgTemp
```

Chapter 3 - Stop! Do These Exercises Before Programming

1. What is in the variable **result** when the calculation is finished?

```
double result = 123 / 10 + .5;
```

Ansr: 12.5

2. What is in the variable **result** when the calculation is finished?

```
double result = 123 / 10. + .5;
```

Ansr: 12.8

3. What is in the variable **result** when the calculation is finished?

```
char a = 2;
short b = 3;
long c = 100000L;
double result = b / a + c;
```

Ansr: 100001.

4. What is in the variable **result** when the calculation is finished?

```
char a = 2;
short b = 3;
long c = 100000L;
double result = (double) b / a + c;
```

Ansr: 100001.5

5. What is in the variable **result** when the calculation is finished?

```
char a = 2;
short b = 3;
long c = 100000L;
double result = b / a + (double) c;
```

Ansr: 100001.

On the next two problems, fix the errors by changing the calculation line; do not change the data types of the variables.

6. Fix the compiler truncation warning message.

```
int itemsOrdered;
double totalCost;
double unitCost;
itemsOrdered = totalCost / unitCost;
```

Ansr: **`itemsOrdered = (int) (totalCost / unitCost);`**

7. Repair the equation so that **totalBytes** contains the correct amount even on old DOS systems.

```
short k = 1024; // 1k bytes = 1024, 1m bytes = 1024k
short numMegs;
long totalBytes;
totalBytes = numMegs * k * k;
```

Ansr: **`totalBytes = numMegs * (long) k * k;`**

8. What is in **sum** and **count** after these instructions complete?

```
int count = 99;
int sum = 10;
sum += (++count)++;
```

Ansr: sum = 110 count = 101

9. What is in **sum** and **count** after these instructions complete?

```
int count = 99;
int sum = 10;
sum *= count++;
```

Ansr: sum = 990 count = 100

10. What is in **sum** and **count** after these instructions complete?

```
int count = 10;
int sum = 99;
sum /= count++;
```

Ansr: sum = 9 count = 11

Chapter 4 — Design Exercises

1. Avoiding a Mostly Working Program.

Programs usually accept some kind of user input. Here we are dealing with numerical input data. When a specific program uses numerical data, the programmer must be alert for particular numerical values which, if entered, cause problems for the algorithm or method that is being used. The programmer must check for possible incorrect values and take appropriate actions. Sometimes the program specifications tell the programmer what to do if those "bad values" are entered; other times, it is left to the good sense of the programmer to decide what to do.

For each of the following, assume that the input instruction has been executed. You are to sketch in pseudocode the rest of the needed instructions.

a. The program accepts a day of the week from one to seven. It displays Sunday when the day is one; Monday, when two; Saturday, when seven.

```
Input a dayNumber
if (dayNumber < 1 || dayNumber > 7)
        cout << "Error: day must be between 1 and 7\n";
else {
        if (dayNumber == 1)
                cout << "Sunday";
        etc...
}
```

b. Housing Cost Program. When buying a house, the seller specifies the length and width of the outside of the home, the number of stories it has and the asking price. This program calculates the actual cost per square foot of real living area within that home. Usually, 25% of the home area is non-liveable, being occupied by doors, closets, garages and so on. Using this program, a buyer can evaluate which home offers the best living area value. Using pseudocode, indicate what should be done to make this program work appropriately for all possible numerical inputs.

```
input the length, width, numberStories and cost
let grossArea = length * width * numberStories
let nonLivingArea = grossArea * .25
let liveableArea = grossArea – nonLivingArea
let realCostPerLiveableFoot = cost / liveableArea
output the realCostPerLiveableFoot
```

A solution: There is a division here. If liveableArea ever is 0, the program crashes with a divide error. Since liveableArea = grossArea - nonLivingArea, and since nonLivingArea =

grossArea * .25, we see that the liveableArea is really

liveableArea = grossArea - grossArea * .25 or grossArea * (1-.25)

Under what circumstances could liveableArea be 0? Only when grossArea is itself 0. How could grossArea be 0? If either length, width or numerStories is 0, then grossArea is 0.

Further, could a house have -2 stories? Or have a length of -42 feet? Thus, we should check each of these for a value that is less than or equal 0.

```
input the length, width, numberStories and cost
if (length <=0 || width <= 0 || numberStories <= 0) {
   cout << "Error: length, width and number of stories cannot be 0 or negative"
else {
        let grossArea = length * width * numberStories
        ...
}
```

2. Comparison of Cereal Prices. Grocery store shoppers are often looking for the best value for their money. For example, a given type of cereal may come in several different sized boxes, each with a different price. A shopper wants to purchase the most cereal they can for the least money; that is, they want the best value for their money. A further complexity arises with coupons. Specific size boxes may have a coupon available to lower the total cost of that box. This program inputs the data for two different boxes and displays which one has the better value (most cereal for the least money). Write the rest of the pseudocode to determine for each box, the actual cost per ounce. Then, display the actual cost per ounce of each box and which is the better value, box1 or box2.

```
Input box1Weight, box1Cost, box1CouponAmount
Input box2Weight, box2Cost, box2CouponAmount
```

A Solution:

```
Input box1Weight, box1Cost, box1CouponAmount
Input box2Weight, box2Cost, box2CouponAmount
if (box1Weight == 0 || box2Weight == 0)
        Display an error message
else do
        let box1CostPerOunce = (box1Cost - box1CouponAmount) / box1Weight
        let box2CostPerOunce = (box2Cost - box2CouponAmount) / box2Weight
        display "box 1 cost per ounce is ", box1CostPerOunce
        display "box 2 cost per ounce is ", box2CostPerOunce
        if (box1CostPerOunce < box2CostPerOunce)
                Display "Box 1 is better"
        else
                Display "Box 2 is better"
end else
```

Chapter 4 — Stop! Do These Exercises Before Programming

1. Given the following variable definitions, what is the result of each of the following test conditions? Mark each result with either a t (for true or 1) or f (for false or 0).

```
     int x = 10, y = 5, z = 42;
__T_ a. if (x > 0)
__T_ b. if (x > y)
__F_ c. if (x == 0)
__F_ d. if (x == z)
__F_ e. if (x + y > z)
__F_ f. if (x / y == z)
__T_ g. if (x > z / y)
__F_ h. if (x > 0 && z < 10)
__T_ i. if (x > 0 && z >= 10)
__T_ j. if (x > 0 || z < 10)
__T_ k. if (x > 0 || z >= 10)
__T_ l. if (x)
__F_ m. if (!x)
```

2. Using the definitions in 1. above, what is the output of the following code?

```
     if (z <= 42)
      cout << "Hello\n";
     else
      cout << "Bye\n";
```

Ansr: Hello

3. Using the definitions in 1. above, what is the output of the following code?

```
     int t = y > x ? z : z + 5;
     cout << t;
```

Ansr: 47

4. Correct all the errors in the following coding. The object is to display the fuel efficiency of a car based on the miles per gallon it gets, its **mpg**.

```
     if (mpg > 25.0) {
       cout << Gas Guzzler\n";
     else
       cout << "Fuel Efficient\n";
```

Ansr:

```
     if (mpg < 25.0)
       cout << "Gas Guzzler\n";
     else
       cout << "Fuel Efficient\n";
```

In the next three problems, repair the If-Then-Else statements. However, maintain the spirit of each type of If-Then-Else style. Do not just find one way to fix it and copy that same "fix" to all three problems. Rather fix each one maintaining that problem's coding style.

5. Correct all the errors in the following coding. The object is to display "equilateral triangle" if all three sides of a triangle are equal.

```
if (s1 == s2 == s3);
   {
   cout << "equilateral triangle\n";
   }
else;
   cout >> "not an equilateral triangle\n";
```

Ansr:

```
if (s1 == s2)
     if (s1 == s3)
        cout << "equilateral triangle\n";
     else
        cout << "not an equilateral triangle\n";
else
     cout << "not an equilateral triangle\n";
```

6. Correct all the errors in the following coding. The object is to display "equilateral triangle" if all three sides of a triangle are equal.

```
if (s1 == s2)
   if (s2 == s3)
      cout << "equilateral triangle\n";
cout >> "not an equilateral triangle\n";
```

Ansr:

```
if (s1 == s2 && s2 == s3)
   cout << "equilateral triangle\n";
else
   cout << "not an equilateral triangle\n";
```

7. Correct all the errors in the following coding. The object is to display "equilateral triangle" if all three sides of a triangle are equal.

```
if (s1 == s2) {
   if (s2 == s3) {
      cout << "equilateral triangle\n";
   }
   else {
      cout >> "not an equilateral triangle\n";
   }
}
```

Ansr:

```
if (s1 == s2) {
   if (s2 == s3) {
      cout << "equilateral triangle\n";
   }
   else {
      cout << "not an equilateral triangle\n";
   }
}
else {
   cout << "not an equilateral triangle\n";
}
```

8. Correct this grossly inefficient set of decisions so that no unnecessary decisions are made.

```
if (day == 1)
   cout << "Sunday\n";
if (day == 2)
   cout << "Monday\n";
if (day == 3)
   cout << "Tuesday\n";
if (day == 4)
   cout << "Wednesday\n";
if (day == 5)
   cout << "Thursday\n";
if (day == 6)
   cout << "Friday\n";
if (day == 7)
   cout << "Saturday\n";
```

Ansr:

```
if (day == 1)
   cout << "Sunday\n";
else if (day == 2)
   cout << "Monday\n";
else if (day == 3)
   cout << "Tuesday\n";
else if (day == 4)
   cout << "Wednesday\n";
else if (day == 5)
   cout << "Thursday\n";
else if (day == 6)
   cout << "Friday\n";
else if (day == 7)
   cout << "Saturday\n";
```

9. Correct this non-optimum solution. Consider all of the numerical possibilities that the user could enter for variable **x**. Rewrite this coding so that the program does not crash as a result of the numerical value entered by the user. You may display appropriate error messages to the user. Ignore the possibility of the user entering in nonnumerical information by accident.

```
double x;
double root;
double reciprocal;
cin >> x;
root = sqrt (x);
reciprocal = 1 / x;
cout << x << " square root is " << root
     << " reciprocal is " << reciprocal << endl;
```

Ansr:

```
double x;
double root;
double reciprocal;
cin >> x;
if (x == 0)
     cout << "Error: x cannot be 0 for a reciprocal\n";
else if (x < 0)
     cout << "Error: x cannot be negative for sqrt\n";
else {
     root = sqrt (x);
     reciprocal = 1 / x;
     cout << x << " square root is " << root
          << " reciprocal is " << reciprocal << endl;
}
```

10. Correct this inherently unsound calculation.

```
double x;
double y;
cin >> x;
y = x * x + 42.42 * x + 84.0 / (x * x * x + 1.);
if (!y || y == x) {
 cout << "x's value results in an invalid state.\n"
 return 1;
}
```

Ansr:

```
double x;
double y;
cin >> x;
if (fabs (x-1.) < .00001)
     cout << "Error: x is too close to 0!!\n";
else {
     y = x * x + 42.42 * x + 84.0 / (x * x * x + 1.);
```

```
        if (!y || y == x) {
          cout << "x's value results in an invalid state.\n";
          return 1;
        }
    }
```

Chapter 5 — Design Exercises

1. Write a short loop that inputs integers from the user and sums them until the user enters any negative number. When the loop ends, display the sum of the numbers and the average value of the numbers that were entered.

A solution:
main storage
num count sum avg

```
sum = count = avg = 0;
cout << "Enter a number or a negative number to quit: ";
while (cin >> num && num > 0) {
     count++;
     sum += num;
}
if (count)
     avg = sum / count;
cout << "Sum = " << sum << "   Average = " << avg << endl;
```

2. The migration flight of a flock of birds has been observed by scientists. From data returned by electronic tracking devices attached to the birds' legs, they have created a file consisting of the number of miles traveled each day by the birds. The first line in **migrate.txt** is the number of days of observations. Each subsequent line consists of the miles traveled that day. For example, if the first number in the file is 30, then there are 30 lines after that first line representing the number of miles traveled each day. Write the pseudocode to input the file and compute the average daily distance the birds have traveled.

A solution:
main storage
number infile miles sum avg j

open infile using migrate.txt file
infile >> number;
j = 0
sum = 0
avg = 0
while (j < number) do the following
    infile >> miles
    sum = sum + miles
    j++
end while

if (number > 0) then avg = sum / number
display avg

Chapter 5 — Stop! Do These Exercises Before Programming

Correct all errors in these programs. The first six illustrate different basic methods of creating the input looping process. However, they contain one or more errors. When you correct them, do NOT just convert them all to a single method — retain the intended input method. In other words, on the first three problems, the **while** loop was intended to not only input the values but also to control the looping process. Problems 4, 5 and 6 use a different method; they should not be rewritten as duplicates of Problem 1.

1. This program is to input pairs of integers from a file called **integers.txt**.

```
#include <iostream>
#include <fstream>
using namespace std;
int main(){
 ifstream infile;
 if (infile) {
  cout << "Error cannot open file integers.txt\n"
  return 1;
 }
 int i, j;
 while (infile << i << j) {
  process them and output results
  ...
 }
 infile.close ();
 return 0;
}
```

Ansr:

```
#include <iostream>
#include <fstream>
using namespace std;
int main(){
 ifstream infile ("integers.txt");
 if (!infile) {
  cout << "Error cannot open file integers.txt\n";
  return 1;
 }
 int i, j;
 while (infile >> i >> j) {
  process them and output results
  ...
 }
 infile.close ();
```

```
 return 0;
}
```

2. This program is to input pairs of integers from a file called **d:\testdata\data.txt**.

```
#include <iostream>
#include <fstream>
using namespace std;
int main(){
 ifstream infile ("d:\testdata\data.txt");
 if (!infile) {
  cout << "Error cannot open file data.txt\n"
  return 1;
 }
 int i, j;
 while (infile >> i >> j) {
  process them and output results
  ...
 }
 infile.close ();
 return 0;
}
```

Ansr:

```
#include <iostream.h>
#include <fstream.h>
int main(){
 ifstream infile ("d:\\testdata\\data.txt");
 if (!infile) {
  cout << "Error cannot open file data.txt\n";
  return 1;
 }
 int i, j;
 while (infile >> i >> j) {
  process them and output results
  ...
 }
 infile.close ();
 return 0;
}
```

3. This program is to input pairs of integers from a file called **inputdata.txt**.

```
#include <iostream>
#include <fstream>
using namespace std;
int main(){
 ifstream infile ("inputdata.txt");
 if (!infile) {
  cout << "Error cannot open file inputdata.txt\n"
  return 1;
 }
 int i, j;
 while (cin >> i >> j) {
  process them and output results
  ...
 }
 infile.close ();
 return 0;
}
```

Ansr:

```
#include <iostream>
#include <fstream>
using namespace std;
int main(){
 ifstream infile ("inputdata.txt");
 if (!infile) {
  cout << "cannot open file inputdata.txt\n";
  return 1;
 }
 int i, j;
 while (infile >> i >> j) {
  process them and output results
  ...
 }
 infile.close ();
 return 0;
}
```

4. This program is to input pairs of integers from a file called **filedata.txt**.

```
#include <iostream>
#include <fstream>
using namespace std;
int main(){
 ifstream infile ("filedata.txt");
 if (!infile) {
  cout << "Error cannot open filedata.txt\n"
  return 1;
```

```
 }
 int i, j;
 infile >> i >> j;
 while (infile) {
  infile  >> i >> j
  process them and output results
  ...
 }
 infile.close ();
 return 0;
}
```

Ansr:

```
#include <iostream>
#include <fstream>
using namespace std;
int main(){
 ifstream infile ("filedata.txt");
 if (!infile) {
  cout << "Error cannot open filedata.txt\n";
  return 1;
 }
 int i, j;
 infile >> i >> j;
 while (infile) {
  process them and output results
  ...
  infile  >> i >> j;      }
 infile.close ();
 return 0;
}
```

5. This program is to input pairs of integers from a file called **filedata.txt**.

```
#include <iostream>
#include <fstream>
using namespace std;
int main(){
 ifstream infile ("filedata.txt");
 if (!infile) {
  cout << "Error cannot open filedata.txt\n"
  return 1;
 }
 int i, j;
 while (cin) {
  process them and output results
  ...
  infile  >> i >> j
```

```
 }
 infile.close ();
 return 0;
}
```

Ansr:

```
#include <iostream>
#include <fstream>
using namespace std;
int main(){
 ifstream infile ("filedata.txt");
 if (!infile) {
  cout << "Error cannot open filedata.txt\n";
  return 1;
 }
 int i, j;
 infile  >> i >> j;
 while (infile) {
  process them and output results
  ...
  infile  >> i >> j;
 }
 infile.close ();
 return 0;
}
```

6. This program is to input pairs of integers from a file called **filedata.txt**.

```
#include <iostream>
#include <fstream>
using namespace std;
int main(){
 ifstream infile ("filedata.txt");
 if (!infile) {
  cout << " Error cannot open filedata.txt\n"
  return 1;
 }
 int i, j;
 while (infile.good());{
  infile  >> i >> j
  process them and output results
  ...
 }
 infile.close ();
 return 0;
}
```

Ansr:

```
#include <iostream>
```

```
#include <fstream>
using namespace std;
int main(){
 ifstream infile (" filedata.txt");
 if (!infile) {
  cout << "Error cannot open filedata.txt\n";
  return 1;
 }
 int i, j;
 infile  >> i >> j;
 while (infile.good()) { // removed errant ;
  process them and output results
  ...
  infile  >> i >> j;
 }
 infile.close ();
 return 0;
}
```

The next four questions refer to this short program. Note the user enters CTL-Z to signal the end of file or input.

```
#include <iostream>
using namespace std;
int main(){
 int i, j;
 while (cin >> i >> j)
  cout << i << " " << j << " ";
 return 0;
}
```

7. What is the output if this is the input: 5 6 7 8 CTL-Z
Ansr: 5 6 7 8

8. What is the output if this is the input: 5 6 A 7 8 CTL-Z
Ansr: 5 6

9. What is the output if this is the input: 1 2 3.4 5 6 CTL-Z
Ansr: 1 2

10. What is the output if this is the input: 1 2 3 A 5 6 CTL-Z
Ansr: 1 2

11. A programmer wrote this program to input five numbers from the user. What is wrong with this program? How can it be fixed?

```
#include <iostream>
using namespace std;
int main(){
 double number;  // number from user
 int    count;   // stop after 5 numbers inputted
 while (count <= 5) {
  cout << "Enter a number: ";
  cin >> number;
  count++;
 }
```

Ansr:

```
#include <iostream>
using namespace std;
int main(){
 double number;  // number from user
 int    count = 0;   // stop after 5 numbers inputted
 while (count < 5) { // removed = sign
  cout << "Enter a number: ";
  cin >> number;
  count++;
 }
```

12. Since the previous version did not work, he rewrote it believing this version would input five numbers from the user. What is wrong with this program? How can it be fixed?

```
#include <iostream>
using namespace std;
int main(){
 double number;  // number from user
 int    count;   // stop after 5 numbers inputted
 for (count=1; count<6; count++) {
  cout << "Enter a number: ";
  cin >> number;
  count++;
 }
```

Ansr:

```
#include <iostream>
using namespace std;
int main(){
 double number;  // number from user
 int    count;   // stop after 5 numbers inputted
 for (count=1; count<6; count++) {
  cout << "Enter a number: ";
  cin >> number;
// remove   count++;
 }
```

13. Since the first two versions did not work, he then tried to write this program to input five numbers from the user. What is wrong with this program? How can it be fixed?

```
#include <iostream>
using namespace std;
int main(){
 double number;  // number from user
 int    count;   // stop after 5 numbers inputted
 do {
  cout << "Enter a number: ";
  cin >> number;
 } while (count < 6);
```

Ansr:

```
#include <iostream>
using namespace std;
int main(){
 double number;  // number from user
 int    count = 1;   // stop after 5 numbers inputted
 do {
  cout << "Enter a number: ";
  cin >> number;
  count++;
 } while (count < 6);
```

14. This program is supposed to write the sum of the odd integers from one to fifty to the file **sum.txt**. What is wrong with it? How can it be fixed?

```
#include <iostream>
using namespace std;
int main(){
 ofstream outfile ("sum.txt");
 if (!outfile) {
  cout << "Error cannot open sum.txt\n"
  return 1;
 }
 int j = 1, sum;
 while (j < 50) {
  sum += j;
  j++;
 }
 cout << sum << endl
 return 0;
}
```

Ansr:

```
#include <iostream>
#include <fstream>
using namespace std;
int main(){
```

```
 ofstream outfile ("sum.txt");
 if (!outfile) {
  cout << "Error cannot open sum.txt\n";
  return 1;
 }
 int j = 1, sum = 0;
 while (j < 50) {
  sum += j;
  j++;
  j++; // or replace these with j+=2;
 }
 outfile << sum << endl;
 return 0;
}
```

15. What is printed by this program?

```
#include <iostream>
#include <iomanip>
using namespace std;
int main(){
 int a = 5;
 while (a) {
  cout << a << endl;
  --a;
 }
 cout << a << endl;
 return 0;
}
```

Ansr:
5
4
3
2
1
0

16. What is printed by this program where the decrement has been moved into the test condition as a postfix decrement?

```
#include <iostream>
#include <iomanip>
using namespace std;
int main(){
 int a = 5;
 while (a--) {
```

```
        cout << a << endl;
       }
       cout << a << endl;
       return 0;
      }
```

Ansr:
4
3
2
1
0
-1

17. What is printed by this program where the decrement has been moved into the test condition as a prefix decrement?

```
      #include <iostream>
      #include <iomanip>
      using namespace std;
      int main(){
       int a = 5;
       while (--a) {
        cout << a << endl;
       }
       cout << a << endl;
       return 0;
      }
```

Ansr:
4
3
2
1
0

Chapter 6 — Design Exercises

1. Acme Company is planning to create a Shapes Helper Package that consists of a number of helpful graphical drawing functions. Write the pseudocode for each of these functions. The rectangle functions are passed a length and width. The circle functions are passed the radius.

getRectangleArea() returns the area of the rectangle.
getRectanglePerimeter() returns the perimeter of the rectangle.
getCircleArea() returns the area of the circle (PI * radius * radius)
getCircleCircum() returns the circumference (2 * PI * radius)

A solution

```
getRectangleArea() is passed length and width
 if (length <0 || width <0)
   cout << "Error: rectangle length and width should not be negative"
 else
    return length * width

getRectanglePerimeter() is passed the length and width
 if (length <0 || width <0)
   cout << "Error: rectangle length and width should not be negative"
 else
    return length * 2 + width * 2

getCircleArea() is passed radius
  let PI = acos(-1.)
  if radius is < 0
    cout << "Error: radius cannot be negative"
  else
    return  PI * radius * radius

getCircleCircum() is passed radius
  let PI = acos(-1.)
  if radius is < 0
    cout << "Error: radius cannot be negative"
  else
    return  2 * PI * radius
```

2. A billing application needs to frequently determine the number of days between a pair of dates. A single date is stored as three integers for month, day and year. The billing program needs to know the number of days between pairs of dates for every set of data in the master file. Thus, it would be desirable for the main program to do something like

```
if (numDays (.....) > 30) { // print a bill
```

In order to calculate the number of days between two dates, it is usually more convenient to convert the date into a Julian day number, which is the number of days since noon on January 1, 4713 B.C. The following sequence converts a date (month, day, year) into the corresponding Julian day.

if the month is 1 or 2, add 12 to month and subtract 1 from the year
Julian day = 1720994.5 + day + (int)((month + 1) * 30.6001) +
(int) (365.25 * year)
if the original date is greater than October 15, 1582, then add one more term to the above Julian day equation (int) (2 – year / 100 + year / 400).

Since the **numDays()** function needs to convert two dates into the Julian day equivalent, it makes sense to also have a **toJulian()** function. Write out the pseudocode for each of the two functions: **numDays()** and **toJulian()**. Be sure to indicate what variables are being passed to your functions and what is being returned.

A solution:

```
toJulian() is passed month, day and year
    origmonth = month
    origyear = year
    if month is 1 or month is 2
       then do
            add 12 to month
            subtract 1 from the year
       end do
    JulianDay = 1720994.5 + day + (int)((month + 1) * 30.6001) +
                (int) (365.25 * year)
    if origyear >= 1582 and origmonth >= 10 and day > 15
            JulianDay += (int) (2 – year / 100 + year/400)
    return JulianDay

numDays() is passed day1, month1, year1 and day2, month2, year2
    julday1 = toJulian (month1, day1, year1)
    julday2 = toJulian (month2, day2, year2)
    return julday1 – julday2
```

Chapter 6 — Stop! Do These Exercises Before Programming

Correct the syntax and logical goofs in the following problems one through seven.

1. The compiler issues an error on this one. Why? How can it be fixed?

```
#include <iostream>
#include <iomanip>
using namespace std;

int main () {
 double x = 99;
 double y;
 y = fun (x);
 cout << y << endl;
 return 0;
}

double fun (double x) {
 return x * x * x;
}
```

Ansr: missing prototype

```
#include <iostream>
#include <iomanip>
using namespace std;

double fun (double x);

int main () {
 double x = 99;
 double y;
 y = fun (x);
 cout << y << endl;
 return 0;
}

double fun (double x) {
 return x * x * x;
}
```

2. The linker issues an error on this one. Why? How can it be fixed?

```
#include <iostream>
#include <iomanip>
using namespace std;
```

```
fun1 (double x);

int main () {
 double x = 99;
 double y;
 y = fun1(x);
 cout << y << endl;
 return 0;
}

double fun1 (double x) {
 return x * x * x;
}
```

Ansr: incorrect return type on prototype, thus the linker cannot find that version of the function

```
#include <iostream>
#include <iomanip>
using namespace std;

double fun1 (double x);

int main () {
 double x = 99;
 double y;
 y = fun1(x);
 cout << y << endl;
 return 0;
}

double fun1 (double x) {
 return x * x * x;
}
```

3. This one does not work right. Why? How can it be fixed?

```
#include <iostream>
#include <iomanip>
using namespace std;

int fun2 (double x);

int main () {
 double x = 99;
 double y;
 fun2 (x);
 cout << y << endl;
 return 0;
}
```

```
int fun2 (double x) {
 return x * x * x;
}
```

ansr: return double but function says int and no use is made of the returned value

```
#include <iostream>
#include <iomanip>
using namespace std;

double fun2 (double x);

int main () {
 double x = 99;
 double y;
 y = fun2 (x);
 cout << y << endl;
 return 0;
}

double fun2 (double x) {
 return x * x * x;
}
```

4. This one creates a compiler error. Why? How can it be fixed?

```
#include <iostream>
#include <iomanip>
using namespace std;

double fun3 (double x);

int main () {
 double x = 99;
 double y;
 y = fun3 (x);
 cout << y << endl;
 return 0;
}

double fun3 (double x) {
 double sum = x * x * x;
}
```

Ansr: nothing is being returned

```
#include <iostream>
#include <iomanip>
using namespace std;

double fun3 (double x);
```

```
int main () {
 double x = 99;
 double y;
 y = fun3 (x);
 cout << y << endl;
 return 0;
}

double fun3 (double x) {
 double sum = x * x * x;
 return sum;
}
```

5. This one does not produce the correct output. Why? How can it be fixed?

```
#include <iostream>
#include <iomanip>
using namespace std;

double fun4 (double x);

int main () {
 double x = 99;
 double y;
 y = double fun4 (double x);
 cout << y << endl;
 return 0;
}

double fun4 (double x) {
 return x * x * x;
}
```

Ansr: invocation is a prototype

```
#include <iostream>
#include <iomanip>
using namespace std;

double fun4 (double x);

int main () {
 double x = 99;
 double y;
 y = fun4 (x);
 cout << y << endl;
 return 0;
}
```

```
double fun4 (double x) {
 return x * x * x;
}
```

6. This one produces the wrong results. Why? How can it be fixed?

```
#include <iostream>
#include <iomanip>
using namespace std;

double fun5 (double x, int power);

int main () {
 double x = 99;
 double y;
 y = fun5 (x);
 cout << y << endl;
 return 0;
}

double fun5 (double x, int power) {
 double ansr = pow (x, power);
 return power;
}
```

Ansr:

```
#include <iostream>
#include <iomanip>
#include <cmath> // for pow function
using namespace std;

double fun5 (double x, int power);

int main () {
 double x = 99;
 double y;
 y = fun5 (x, 2); // needs a power
 cout << y << endl;
 return 0;
}

double fun5 (double x, int power) {
 double ansr = pow (x, power);
 return ansr; // returning the wrong result
}
```

7. This one creates a compiler error about local functions not supported. Why? How can it be fixed?

```
#include <iostream>
#include <iomanip>
using namespace std;

double fun6 (double x);

int main () {
 double x = 99;
 double y;
 y = fun6 (x);
     double fun6 (double x) {
      return = x * x * x;
     }
 cout << y << endl;
 return 0;
}
```

Ansr: function bodies cannot be inside any other block of coding

```
#include <iostream>
#include <iomanip>
using namespace std;

double fun6 (double x);

int main () {
 double x = 99;
 double y;
 y = fun6 (x);
 cout << y << endl;
 return 0;
}

double fun6 (double x) {
 return = x * x * x;
}
```

8. What will the **cout** display for the variables **a**, **b** and **c**?

```
#include <iostream>
#include <iomanip>
using namespace std;

int  fun (int a, int b);

int main () {
```

```
 int a = 1;
 int b = 2;
 int c = 3;
 c = fun (a, b);
 cout << c << " " << b << " " << a << endl;
 return 0;
}

int  fun (int a, int b) {
 a = 42;
 b = 42;
 return 42;
}
```

Ansr: 42 2 1

9. What will the **cout** display for the variables **a**, **b** and **c**?

```
#include <iostream>
#include <iomanip>
using namespace std;

int  fun (int a, int b);

int main () {
 int a = 1;
 int b = 2;
 int c = 3;
 if (a == 1) {
  int d = 42;
  int c;
  c = fun (d, a);
 }
 cout << c << " " << b << " " << a << endl;
 return 0;
}

int  fun (int a, int b) {
 return a + b;
}
```

Ansr: 3 2 1

Chapter 7 — Design Exercises

1. Design an function called **inputData()**. From the passed input file stream, it is to extract a set of data. The data include the employee Id number (9 digits long), the employee's age in years, and the date that the employee was hired. The hired date is input in the form of mm:dd:yyyy. However, if the year is less than 100, it is not "year 2000 compliant." In that case, add 1900 to the year. The user expects to use the function as follows.

```
while (inputData (infile, id, age, month, day, year)) {
```

A solution:

```
istream& inputData (istream& infile, long&id, int& age,
                    int& month, int& day, int& year) {
     char c;
     infile >> dec >> id >> age >> month >> c
            >> day >> c >> year;
     if (infile) {
          if (year < 100) year += 1900;
     }
     return infile;
}
```

2. Design a function called **circleProperties()**. It is passed the circle's radius. It must calculate the area of the circle and the circumference of the circle. However, if the radius is within .001 from being 0, then return **false** along with zeros for the two properties. If the radius is not 0, then return **true**. The calling function expects to use it similar to this.

```
if (circleProperties (radius, circum, area)) {
```

A solution:

```
bool circleProperties (double radius, double& circum,
                       double& area) {
     if (fabs (radius) < .001) {
          circum = area = 0;
          return false;
     }
     const double PI = acos(-1.);
     circum = 2 * PI * radius;
     area = PI * radius * radius;
     return true;
}
```

Chapter 7 — Stop! Do These Exercises Before Programming

1. Consider the following program to input a date in the format of mm-dd-yyyy, such as `10-22-2001`. What happens if the user inputs by accident `10 22-2001`? What happens if the user enters `10-22 2001`? What happens if the user enters `10a22a2001`?

```
int month, day, year, c;
cin >> month >> c >> day >> c >> year;
```

Ansr: 10 22-2001? 10 into month, 2 into c, 2 into day, - into c, 2001 into year
10-22 2001? 10 into month, - into c, 22 into day, 2 into c, 001 into year
10a22a2001? 10 into month, a into c, 22 into day, a into c, 2001 into year

What would be the results of these three user entries if the input operation had been coded this way?

```
cin >> month;
cin.get (c);
cin >> day;
cin.get (c);
cin >> year;
```

Ansr: 10 22-2001? 10 into month, b into c, 22 into day, - into c, 2001 into year
10-22 2001? 10 into month, - into c, 22 into day, b into c, 2001 into year
10a22a2001? 10 into month, a into c, 22 into day, a into c, 2001 into year

2. Why does this program create a linker error when building the program? How should it be fixed?

```
#include <iostream>
using namespace std;

int sumFunction (int& counter);
int main () {
 int x = 42;
 int sum = sumFunction (x);
 return 0;
}

int sumFunction (int counter) {
 int j;
 int sum = 0;
 for (j=0; j<counter; j++)
  sum += j;
 return sum;
}
```

Ansr: prototype differs from the function and linker cannot find a function that takes an int;

counter is not being changed, so remove the &

```
int sumFunction (int counter);
int main () {
```

3. What is wrong with the first **if** statement? How can it be made to work correctly so that **result** contains the correct value, ignoring leap year troubles?

```
#include <iostream>
using namespace std;
const int ERROR = -1;
int validateDate (int month, int day, int year);
int main () {
 int mon = 10;
 int day = 32;
 int yr = 2000;
 int result;
 if (result = validateDate (mon, day, yr) == ERROR)
  cout << "Bad Date\n"
 ...
int validateDate (int m, int d, int y) {
 if (m < 1 || m > 12 || d < 1 || d > 31 || y < 1900)
   return ERROR;
 else
  return 1;
}
```

Ansr: result holds the t/f result of the test and a ; missing

```
 if ( (result = validateDate (mon, day, yr)) == ERROR)
  cout << "Bad Date\n";
```

4. When this program is executed, why do very unpredictable actions occur? How can it be fixed so that it works properly?

```
#include <iostream>
using namespace std;

void inputDate (iostream in, int& month, int& day,
                int& year);
int main () {
 int month, day, year, quantity;
 double cost;
 inputDate (cin, month, day, year);
 cin >> quantity >> cost;
 ...
void inputDate (iostream in, int& month, int& day,
                int& year) {
 char c;
 in >> month;
```

```
 in.get (c);
 in >> day;
 in.get (c);
 in >> year;
}
```

Ansr: iostreams must be passed by refernce cause they change as data is I/Oed - also it is istream not iostream

```
#include <iostream>
using namespace std;

void inputDate (istream& in, int& month, int& day,
                int& year);
int main () {
 int month, day, year, quantity;
 double cost;
 inputDate (cin, month, day, year);
 cin >> quantity >> cost;
 ...
void inputDate (iostream& in, int& month, int& day,
                int& year) {
 char c;
 in >> month;
 in.get (c);
 in >> day;
 in.get (c);
 in >> year;
}
```

5. When this program is executed, why do very unpredictable actions occur? How can it be fixed so that it works properly?

```
#include <iostream>
using namespace std;

iostream inputDate (iostream& in, int& month, int& day,
                    int& year);
int main () {
 int month, day, year, quantity;
 double cost;
 while (inputDate (cin, month, day, year)) {
  cin >> quantity >> cost;
 ...
iostream inputDate (iostream& in, int& month, int& day,
                    int& year) {
 char c;
 in >> month;
 in.get (c);
```

```
 in >> day;
 in.get (c);
 in >> year;
 return in;
}
```

Ansr: it is istream not iostream; and should return a reference to the stream

```
#include <iostream>
using namespace std;

istream& inputDate (istream& in, int& month, int& day,
                    int& year);
int main () {
 int month, day, year, quantity;
 double cost;
 while (inputDate (cin, month, day, year)) {
  cin >> quantity >> cost;
 ...
istream& inputDate (istream& in, int& month, int& day,
                    int& year) {
```

6. When this program runs, why does **main()**'s **cout** produce erroneous results? How can it be fixed?

```
#include <iostream>
using namespace std;

void inputDate ();
int month, day, year;
int main () {
 int month, day, year;
 inputDate ();
 cout << month << '-' << day << '-' << year;
 ...
void inputDate () {
 char c;
 in >> month >> c >> day >> c >> year;
}
```

Ansr: in inputDate in is not defined; in main, local copies of the variables hide the global ones

```
#include <iostream>
using namespace std;

void inputDate ();

int month, day, year;

int main () {
```

```
// remove  int month, day, year;
      inputDate ();
      cout << month << '-' << day << '-' << year;
      ...
     void inputDate () {
      char c;
      cin >> month >> c >> day >> c >> year;
     }
```

7. When this program runs, why does **main()**'s **cout** produce erroneous results? How can it be fixed?

```
     #include <iostream>
     using namespace std;

     void inputDate ();
     int month, day, year;
     int main () {
      int month, day, year;
      inputDate ();
      cout << month << '-' << day << '-' << year;
      ...
     void inputDate () {
      int month, day, year;
      char c;
      in >> month >> c >> day >> c >> year;
     }
```

Ansr: data is input into local copies in inputDate, in is not defined, main's local variable also hide globals

```
     #include <iostream>
     using namespace std;

     void inputDate ();

     int month, day, year;

     int main () {
// remove  int month, day, year;
      inputDate ();
      cout << month << '-' << day << '-' << year;
      ...
     void inputDate () {
// remove  int month, day, year;
      char c;
      cin >> month >> c >> day >> c >> year;
     }
```

8. This **outputDate()** function is badly designed. Why? How can it be repaired?

```
#include <iostream>
#include <iomanip>
using namespace std;

ostream& outputDate (ostream& out, int& m, int& d, int& y);

int main () {
 int month, day, year;
 ...
 outputDate (cout, month, day, year);
 ...
}

ostream& outputDate (ostream& out, int& m, int& d, int& y){
 out << setw (2) << setfill ('0') << m << '-'
     << setw (2) << d << '-' << setw (4) << y
     << setfill (' ');
 return out;
}
```

Ansr: outputDate is not changing the m, d, or y values, so they should NOT be passed by reference

```
ostream& outputDate (ostream& out, int m, int d, int y);
...
ostream& outputDate (ostream& out, int m, int d, int y){
```

9. When this program is run, a strange, often large, number appears after the correct date on the screen. Why? How can it be removed?

```
#include <iostream>
#include <iomanip>
using namespace std;

ostream& outputDate (ostream& out, int& m, int& d, int& y);

int main () {
 int month, day, year;
 ...
 cout << outputDate (cout, month, day, year) << " "
      << setw (10) << quantity;
 ...
}

ostream& outputDate (ostream& out, int& m, int& d, int& y){
 out << setw (2) << setfill ('0') << m << '-'
     << setw (2) << d << '-' << setw (4) << y
     << setfill (' ');
```

```
 return out;
}
```

Ansr: the main function cout << outputDate displays the memory address of the ostream! Also don't pass m, d, y by reference.

```
#include <iostream>
#include <iomanip>
using namespace std;

ostream& outputDate (ostream& out, int m, int d, int y);

int main () {
 int month, day, year;
 ...
 outputDate (cout, month, day, year) << " " << setw (10)
         << quantity;
 ...
}

ostream& outputDate (ostream& out, int m, int d, int y){
 out << setw (2) << setfill ('0') << m << '-'
     << setw (2) << d << '-' << setw (4) << y
     << setfill (' ');
 return out;
}
```

10. What will **cout** display for the variable **c**? for **b**? for **a**?

```
#include <iostream>
#include <iomanip>
using namespace std;

int  fun (int a, int &b);

int main () {
 int a = 1;
 int b = 2;
 int c = 3;
 c = fun (a, b);
 cout << c << " " << b << " " << a << endl;
 return 0;
}

int  fun (int a, int &b) {
 a = 42;
 b = 42;
 return 42;
```

```
}
```
Ansr: 42 42 1

11. What are the contents of **main()**'s variables **a** and **b** after the first call to **fun()**? What are the contents of **a** and **b** after the second call to **fun()**?

```
#include <iostream>
using namespace std;

void fun (double &x, double y);

int main() {
 double a = 1, b = 2;
 fun (a, b);
 fun (b, a);
}

void fun (double &x, double y) {
 x += y;
 return;
}
```
Ansr: 1st: 3 2
2nd: 3 5

12. What values are printed for **x**, **y** and **z** in **some_fun()** the first time that function is called? What values are printed the second time it is called?

```
#include <iostream>
#include <iomanip>
using namespace std;

void some_fun ();

int x = 1;

int main(){
 int y = 2;
 static int z = 3;
 some_fun();
 x +=10;
 y +=10;
 z +=10;
 some_fun();
 return 0;
}
void some_fun () {
 int y = 2;
```

```
 static int z = 3;
 cout << x << " " << y << " " << z << endl;
 x++;
 y++;
 z++;
}
```

Ansr: 1st: 1 2 3
2nd: 12 2 4

Chapter 8 — Design Exercises

1. Design a Weekly Sales Summary Program

Acme Music Company handles violins, cellos, drums, guitars and pianos. When the weekly sales summary report program is run, the sales file is input and summarized.

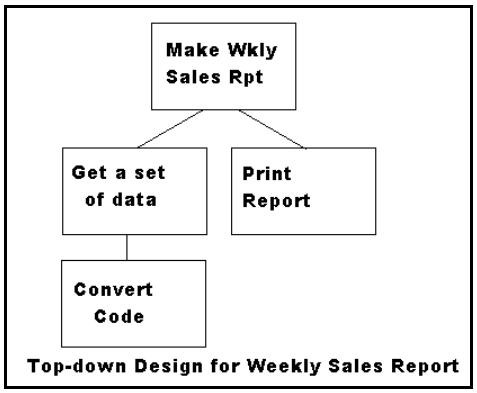
Each line in the sales file represents sales in which the customer purchased an item that was in the store. A line contains the quantity sold, its cost, its id number, a letter code indicating which kind of instrument it is, and finally a nine-digit customer id number. This letter code consists of one of these letters: V, C, D, G, and P – taken from the first letter of each musical instrument type

The summary report should accumulate the total sales of each instrument type. When the end of the file is reached, display a summary of the sales for each instrument type. The report looks like this.

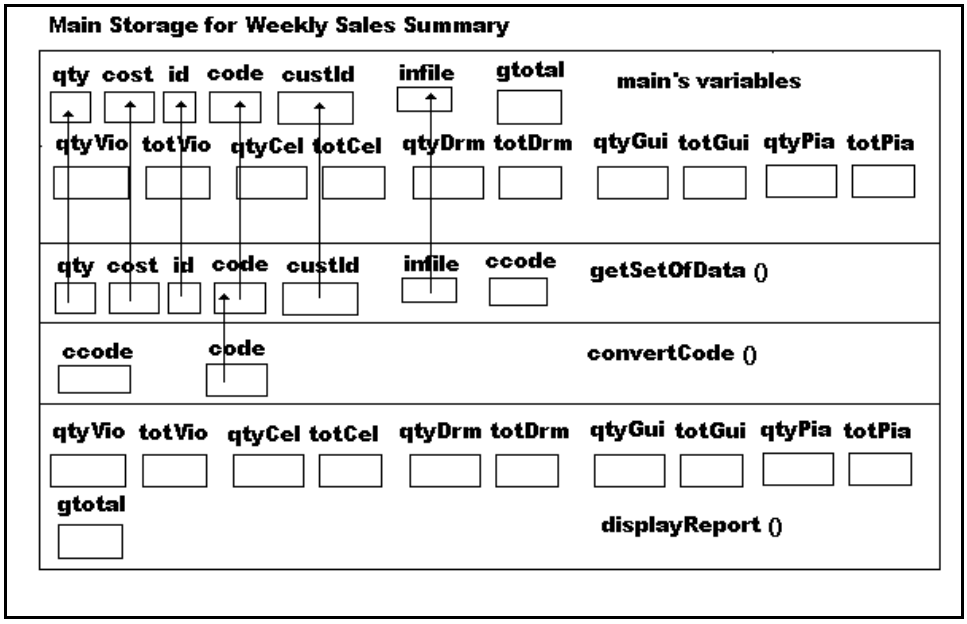
```
     Acme Music Company
     Weekly Sales Summary

Instrument Quantity    Total
             Sold      Sales
  Violin      999    $9,999.99
  Cello       999    $9,999.99
  Drum        999    $9,999.99
  Guitar      999    $9,999.99
  Piano       999    $9,999.99
                     ---------
                    $99,999.99
```

Design the program. First, do a Top-Down Design of the problem to isolate what functions are needed. Next, decide upon whether or not to use an **enum** — that is, how you wish to handle the instrument codes. Then, layout main storage for your functions. Finally, pseudocode the functions needed.

A Solution:



Top-down Design for Weekly Sales Report


Main Storage for Weekly Sales Summary
qty cost id code custId infile gtotal
main's variables
qtyVio totVio qtyCel totCel qtyDrm totDrm qtyGui totGui qtyPia totPia
qty cost id code custId infile ccode
getSetOfData ()
ccode code
convertCode ()
qtyVio totVio qtyCel totCel qtyDrm totDrm qtyGui totGui qtyPia totPia
gtotal
displayReport ()

```
main:
        enum Instrument {Violin, Cello, Drum, Guitar, Piano};
        open infile
        while (getSetOfData (infile, qty, cost, id, code, custId)) {
                switch (code) {
                        case Violin:
                                Add qty to qtyVio
                                Add cost*qty to totVio
                                break
                        case Violin:
                                Add qty to qtyCel
                                Add cost*qty to totCel
                                break
                        case Violin:
                                Add qty to qtyDrm
                                Add cost*qty to totDrm
                                break
                        case Violin:
                                Add qty to qtyGui
                                Add cost*qty to totGui
                                break
                        case Violin:
                                Add qty to qtyPia
                                Add cost*qty to totPia
                                break
```

```
                }
        }
        gtotal = totVio + tot Cel + totDrm + totGui + totPia
        displayReport (qtyVio, totVio, qtyCel, totCel, qtyDrm, totDrm, qtyGui, totGui,
                        qtyPia, totPia, gtotal)
        close infile

getSetOfData is passed a reference to these: infile, qty, cost, id, code, custId
        infile >> qty >> cost >> id >> ccode >> custId;
        if (infile)
                convertCode (ccode, code)
        return infile

convertCode is passed ccode and a reference to code
        ccode = toupper (ccode)
        switch on ccode
                case 'V': code = Violin; break;
                case 'C': code = Cello; break;
                case 'D': code = Drum; break;
                case 'G': code = Guitar; break;
                case 'P': code = Piano; break;
                default: display error and exit
        end switch

displayReport is passed qtyVio, totVio, qtyCel, totCel, qtyDrm, totDrm, qtyGui, totGui,
                        qtyPia, totPia, gtotal
        display heading lines Acme Music Company and Weekly Sales Summary
        display column heading lines Instrument Quantity Total and Sold Sales
        set up for floating point numbers with 2 decimal places
        display "Violin", qtyVio, '$', totVio;
        display "Cello", qtyCel,  '$', totCel;
        display "Drum", qtyDrm,  '$', totDrm;
        display "Guitar", qtyGui,  '$', totGui;
        display "Piano", qtyPia,  '$', totPia;
        display dash line
        display  '$', gtotal
```

2. Design a **petTypeConversion()** function

Acme Pet Shop handles many kinds of pets. Each is identified in many of their data files by a single letter, either upper or lowercase. Having heard about **enum**s, management has decided to convert all of their programs over to using a **Pet enum**. A single function is to be written that all the programs can use to handle this process. The **Pet enum** handles the following: dog, cat, bird, rat, hamster, snake, and gerbil. The letter codes consist of the first letter of the animal types.

The **petTypeConversion()** function is passed four parameters. The first parameter is a **bool** called **which** that indicates which way the conversion is to go. If **which** is **true**, convert the second parameter, a **char** pet code letter, into the appropriate **enum** value and store it in the third parameter which is a reference to a **Pet enum** called **pet**. If **which** is **false**, then display the English word, such as dog, for the passed **Pet enum**, **pet**. The display is done using the fourth parameter, a reference to an **ostream**, **outfile**.

A solution:

```
enum Pet {Dog, Cat, Bird, Rat, Hamster, Snake, Gerbil};

petTypeConversion() passed which, petLetter and a reference to both pet and outfile
    if (which) {
        petLetter = toupper(petLetter);
        switch (petLetter) {
            case ‘D’:
                pet = Dog; break;
            case ‘C’:
                pet = Cat; break;
            case ‘B’:
                pet = Bird; break;
            case ‘R’:
                pet = Rat; break;
            case ‘H’:
                pet = Hamster; break;
            case ‘S’:
                pet = Snake; break;
            case ‘G’:
                pet = Gerbil; break;
            default:
                cerr << “Error bad pet code: “ << petLetter
        }
    }
    else {
```

```
                switch (pet) {
                        case Dog: outfile << “Dog”; break;
                        case Cat: outfile << “Cat”; break;
                        case Bird: outfile << “Bird”; break;
                        case Rat: outfile << “Rat”; break;
                        case Hamster: outfile << “Hamster”; break;
                        case Snake: outfile << “Snake”; break;
                        case Gerbil: outfile << “Gerbil”; break;
                        case Dog: outfile << “Dog”; break;
                        default: cerr << “Error bad pet value”
                }
        }
}
```

Chapter 8 — Stop! Do These Exercises Before Programming

1. The following program segment does not produce the correct results. Why? How can it be fixed?

```
char quantity;
double cost;
cin >> quantity >> cost;
cout << "Total is " << quantity * cost << endl;
```

Ansr: it assumes an ASCII letter is being extracted. Change to **short** or **int**.

short quantity;

2. Acme Department Store has new product information stored in a file whose lines consist of product id, product type and cost. The product id is a **long** integer while the cost is a **double**. The product type is a letter code indicating the category of merchandise, such as A (automotive), C (clothing), S (sports) and so on. A typical line looks like this with one blank on either side of the product type letter.

```
23455 A 4.99
```

However, since these are new items, sometimes the type of product has not yet been determined and that field is blank in that line. The programmer wrote the following input function. It does not work. Why? What can be done to fix it up so that it properly inputs the data whether or not the product type is temporarily blank?

```
istream& getData (istream& infile, long& id, char& type,
                  double& cost) {
 infile >> id >> type >> cost;
 return infile;
}
```

Ansr: extraction skips over ws, thus the blank is skipped.

```
char c;
infile >> id;
infile.get(c);
infile.get(type);
infile >> cost;
```

3. Another Acme programmer attempted to fix the program in Problem 2 above by coding the following function. It does not work properly either. Why? How could it be fixed to work correctly?

```
istream& getData (istream& infile, long& id, char& type,
                  double& cost) {
 infile >> id;
 infile.get (type);
 infile >> cost;
 return infile;
```

```
}
```

Ansr: this inputs the leading blank before the type letter or blank as the type field

```
char c;
infile >> id;
infile.get(c);
infile.get(type);
infile >> cost;
```

4. What is wrong with the following Do Case coding? How can it be fixed up so that it would work?

```
double month;
switch (month) {
 case 1:
 case 2:
 case 12:
   // winter costs are 25% higher
   double sum;
   sum = qty * cost * 1.25;
   break;
 default:
   double sum;
   sum = qty * cost;
   break;
}
cout << sum;
```

Ansr: variables cannot be defined within a case stmt; cannot switch on doubles

```
int month;
double sum;
switch (month) {
 case 1:
 case 2:
 case 12:
   // winter costs are 25% higher
   sum = qty * cost * 1.25;
   break;
 default:
   sum = qty * cost;
   break;
}
cout << sum;
```

5. The programmer goofed while coding this Do Case to calculate the shift bonus for the employee payroll. What is wrong and how can it be fixed?

```
char shift;
switch (shift) {
```

```
 case '1':
  pay = hours * rate;
 case '2':
  pay = hours * rate * 1.05;
 case '3':
  pay = hours * rate * 1.12;
}
cout << pay;
```

Ansr: forgot the break stmts

```
char shift;
switch (shift) {
 case '1':
  pay = hours * rate; break;
 case '2':
  pay = hours * rate * 1.05; break;
 case '3':
  pay = hours * rate * 1.12; break;
}
cout << pay;
```

6. The programmer wanted to setup an enumerated data type to handle the employee's shift. However, the following coding fails. Why? How can it be repaired?

```
Enum ShiftType = First, Second, and Third;
```

Ansr: watch case and syntax

enum ShiftType {First, Second, and Third};

7. A programmer setup the following **enum** to handle the product types.

```
enum ProductTypes {Games, Auto, Clothing, Appliances};
```

In the input a set of data function, the user is instructed to enter a letter for the product type: G, A, C or A. What is the design flaw and why does not the following input coding work? How can it be repaired?

```
ProductTypes prodType;
infile >> prodType;
```

Ansr: there are 2 letter codes the same, A; change Appliances to say the letter P
you cannot input enum variables, you need to input a char and convert

```
ProductTypes prodType;
char c;
infile >> c;
c = toupper (c);
switch (c) {
 case 'G': prodType = Games; break;
 case 'A': prodType = Auto; break;
 case 'C': prodType = Clothing; break;
 case 'P': prodType = Appliances; break;
```

```
}
```

8. In Problem 7, the programmer got frustrated and then did the following which does not compile. Why? Can this coding be repaired?

```
ProductTypes prodType;
char c;
infile >> c;
if (c == 'A')
 prodType = 1;
if (c == 'G')
 prodType = 0;
```

Ansr: cannot assign integer values to enum variables, only the enum data values

```
if (c == 'A')
 prodType = Auto;
if (c == 'G')
 prodType = Games;
```

Chapter 9 — Design Exercises

1. Mail Sorting Statistics

Design a program that displays mail sorting statistics for the Post Office. At the end of each worker's shift at the mail sorting room, that mail sorter's id and the number of pieces of mail they have sorted are appended to the **DailyStats.txt** file. At the end of the day, this program is run to produce the Mail Sorting Stats Report.

```
The Average Number Sorted Today: 999999

Sorter   Number   Percentage
  Id     Sorted   of Average
 9999     99999    999.99%
 9999     99999    999.99%
 9999     99999    999.99%
```

Allow for a maximum of 100 workers. The percentage of the average sorted is given by that workers number sorted * 100 / average number sorted.

A solution:
design: main calls LoadArrays, FindAvg, ComputePercents, and DisplayReport
main defines
const int MAX = 100;
long id[MAX];
long numSorted[MAX];
long num = LoadArray (id, numSorted, MAX);
long avg = FindAvg (numSorted, num);
double percents[MAX];
ComputePercentages (numSorted, num, avg, percents);
DisplayReport(id, numSorted, percents, num, avg);
LoadArray is passed the arrays id and numSorted along with the limit
open infile using DailyStats.txt file
j = 0;
while (j<num && infile >> id[j] >> numSorted[j]) j++;
close infile
return j;
FindAvg is passed the array numSorted and the num in it
sum = 0;
for (j=0; j<num; j++)
sum += numSorted[j];
return sum / num
ComputePercentages is passed the arrays numSorted and percents and the num in them and the avg that was sorted

for (j=0; j<num; j++) {
percents[j] = numSorted[j] * 100 / avg
}
DisplayReport is passed the arrays id, numSorted and percents, the num in the arrays and the avg
display heading line and avg
display column heading two lines
for (j=0; j<num; j++) {
display id[j], numSorted[j] and percents[j] and a % sign
}

2. The Optimum Hours

A fast food restaurant has installed a traffic counter to count the number of cars driving by its store. It logs a count of the number of cars in each ten-minute period from 8am to 10pm. Each line contains a time of the form of hh:mm followed by the count of the cars passing during that interval. The program should first load the file **counts.txt** into three arrays, hour, minute, count. Find the average number of cars per ten-minute interval throughout the day. Now display that average count and the three highest counts that were observed along with their time. The report appears similar to this.

```
Summary of Passing Cars
Average count: 9999
hh:mm   count
99:99     9999
99:99     9999
99:99     9999
```

Based upon these results, management will ensure that more employees are working during the potential higher traffic times.

A Solution:
the design: main calls LoadArray, ComputeAvg, SortArray, and DisplayReport
main variables
const int MAX = 1000;
short hour[MAX];
short minute[MAX];
long count[MAX];
long num = LoadArray (hour, minute, count, MAX);
long avg = ComputeAverage(count, num);
SortArrays(hour, minute, count, num);
DisplayReport (hour, minute, count, num, avg);
LoadArray is passed the arrays hh, mm, count, and the limit
open infile using file counts.txt
let j = 0;

```
        define c as a char for the :
        while (j<limit && infile >> hh[j] >> c >> mm[j] >> count[j]) j++;
        close infile
        return j
ComputeAvg is passes the array count and the num in it
        sum = 0;
        for (j=0; j<num; j++)
                sum += count[j];
        return sum / num;
SortArrays is passed the arrays hh, mm and count along with the num in them
        for (j=0; j<num-1; j++) {
                for (k=j+1; k<num; k++) {
                        if (count[k] > count[j]) {
                                temp = count[k];
                                count[k] = count[j];
                                count[j] = temp;
                                temphh = hh[k];
                                hh[k] = hh[j];
                                hh[j] = temphh;
                                tempmm = mm[k];
                                mm[k] = mm[j];
                                mm[j] = tempmm;
                        }
                }
        }
DisplayReport is passed the avg and the arrays hh, mm and count with the num in them
        display title line
        display avg count msg and then avg
        display column headings
        numToDo = num < 3 ? num : 3;
        for (k=0; k<numToDo; k++) {
                display hh[k], mm[k], count[k]
        }
```

Chapter 9 — Stop! Do These Exercises Before Programming

1. This attempt to input the five tax rates for the states in which ACME sells products won't compile. Why? How can it be repaired so that it works?

```
int main () {
 double taxrates[5];
 cin >> taxrates;
 ...
```

Ansr: you cannot input a whole array as a single item, input individual elements

```
int main () {
 double taxrates[5];
 for (int k = 0; k < 5; k++)
    cin >> taxrates[k];
 ...
```

2. This attempt also does not work properly, though it compiles. Why? How can it be fixed?

```
int main () {
 double taxrates[5];
 for (int j=1; j<6; j++)
  cin >> taxrates[0];
 ...
```

Ansr: subscripts range from 0 to 4 if there are 5 in the array, and they are all going into element 0

```
int main () {
 double taxrates[5];
 for (int j=0; j<5; j++)
  cin >> taxrates[j];
 ...
```

3. This attempt compiles fine, but at run time it crashes. Why? How can it be fixed so that it works properly?

```
int main () {
 double taxrates[5];
 for (int j=1; j<6; j++)
  cin >> taxrates[j];
 ...
```

Ansr: core overlay as subscript is out of range

```
int main () {
 double taxrates[5];
 for (int j=0; j<5; j++)
  cin >> taxrates[j];
 ...
```

4. Now that the tax rates have been correctly input, a major design flaw surfaced. If the main program inputs a state code, such as 13 (for Illinois), how can the corresponding tax rate from the array of five tax rates be found? A file was created in which each input line of the **taxrates.txt** file contains the integer state code and the corresponding tax rate. This attempt at making a function to load the arrays fails to work properly. Why? How can it be fixed?

```
int main () {
 double taxrates[5];
 int    states[5];
 loadArray (states[], taxrates[5]);
 ...
void loadArray (int states[], double taxrates) {
 ifstream infile ("taxrates.txt");
 int j=0;
 while (infile >> states[j] >> taxrates) {
  j++);
 }
 infile.close();
}
```

Ansr: taxrates must be an array in loadArray, function not invoked properly

```
int main () {
 double taxrates[5];
 int    states[5];
 loadArray (states, taxrates);
 ...
void loadArray (int states[], double taxrates[]) {
 ifstream infile ("taxrates.txt");
 int j=0;
 while (infile >> states[j] >> taxrates[j]) {
  j++);
 }
 infile.close();
}
```

5. Since the previous attempt to make the load function failed, the programmer threw the whole thing away and started over. This is the next attempt, also doomed, though the programmer finally got it to nearly compile. Why? How can it be fixed up?

```
const int MAX = 5
int main () {
 double taxrates[MAX];
 int    states[MAX];
 loadArray (states[MAX], taxrates[MAX], MAX);
 ...
void loadArray (int states, double taxrates, int MAX) {
 ifstream infile ("taxrates.txt");
 int j=0;
```

```
 while (infile >> states >> taxrates && j < MAX) {
  j++);
 }
 infile.close();
}
```

Ansr: improper invocation of the function; function must have arrays passed etc

```
const int MAX = 5
int main () {
 double taxrates[MAX];
 int    states[MAX];
 loadArray (states, taxrates, MAX);
 ...
void loadArray (int states, double taxrates, int max) {
 ifstream infile ("taxrates.txt");
 int j=0;
 while (j < max && infile >> states[j] >> taxrates[j]) {
  j++);
 }
 infile.close();
}
```

6. Undaunted by his previous difficulties, the programmer decided that a **matchState()** function was needed. Here is the first attempt. It compiles but does not work. Why? How can you fix it so that it does work properly?

```
int matchState (int states[], int state) {
 int j = 0;
 while (state != states[0])
  j++;
 }
 return j;
}
```

Ansr: always checking the current state to the first element in the states array - if no match, no j to return and it runs off the end of the array...

```
const int NOMATCH = -1;
int matchState (int states[], int state, int numStates) {
 int j = 0;
 while (j < numStates) {
  if (state == states[j]) return j;
  j++;
 }
 return NOMATCH;
}
```

7. With a working way to find the matching state, work began on the main function's

calculation's loop. This is what the programmer produced. It fails to work properly. Why? How can it be repaired?

```
int    statecodes[5];
double taxrates[5];
double cost;
int    quantity;
int    statecd;
double tax;
double total;
double grandTotal;
int    matchingStateCodeSubscript;
while (infile2 >> quantity >> cost >> statecd) {
 total = quantity * cost;
 matchState (statecodes, statecd);
 tax = total * taxrates[matchingStateCodeSubscript];
 grandTotal = total + tax;
 cout << grandTotal << endl;
}
```

Ansr: no use was made of the returned subscript so matchingStateCodeSubscript has no value

```
...
int    matchingStateCodeSubscript;
while (infile2 >> quantity >> cost >> statecd) {
 total = quantity * cost;
 matchingStateCodeSubscript =
           matchState (statecodes, statecd);
 tax = total * taxrates[matchingStateCodeSubscript];
 grandTotal = total + tax;
 cout << grandTotal << endl;
}
```

Further, if there is no match, what subscript should be used??

8. A programmer was asked to make a program to convert students' final raw grades into a letter grade. The specifications called for the use of arrays. One array holds the lowest possible score for a particular letter grade and the other array holds the corresponding letter grade. The programmer produced the following coding. While it works, his boss immediately ordered a revision and told him to just initialize the arrays not assign them values at run time. How can this be done?

```
int main () {
 int  rawScores[5];
 char grades[5];
 rawScores[0] = 90;
 grades[0] = 'A';
 rawScores[1] = 80;
 grades[1] = 'B';
 rawScores[2] = 70;
```

```
      grades[3] = 'C';
      rawScores[3] = 60;
      grades[3] = 'D';
      rawScores[4] = 0;
      grades[4] = 'F';
```

Ansr:

```
     int main () {
      int  rawScores[5] = {90, 80, 70, 60, 0};
      char grades[5] = {'A', 'B', 'C', 'D', 'F'};
```

9. Next, having gotten the two arrays loaded properly, the programmer proceeded to write the code to convert students' raw scores into letter grades. The students complained bitterly about the results. Desk check with a grade of 94.78. What is wrong with this and how can it be fixed to work properly?

```
     int main () {
      int rawScores[5]....
      char grade[5]....
      double rawscore;
      long idNum;
      char grade;
      while (cin >> idNum >> rawscore) {
       for (int j=4; j>=0; j++)
        if (rawscore > rawScores[j]) break;
       }
       grade = grades[j];
       cout << idNum << " " << grade << endl;
      }
```

Ansr: a 94 is certainly greater than 0

```
     int main () {
      int rawScores[5]....
      char grade[5]....
      double rawscore;
      long idNum;
      char grade;
      while (cin >> idNum >> rawscore) {
       for (int j=0; j<5; j++)
        if (rawscore >= rawScores[j]) break;
       }
       grade = grades[j];
       cout << idNum << " " << grade << endl;
      }
```

10. Hastily the programmer spotted his errors and recoded the program as follows. This is now a "mostly working" program. Far fewer students were complaining about their grades. Can you

spot the remaining error? One of the students complaining bitterly about their grade received a raw score of 89.997. How can the program now be fixed?

```
int main () {
 int rawScores[5]....
 char grade[5]....
 double rawscore;
 long idNum;
 char grade;
 while (cin >> idNum >> rawscore) {
  for (int j=0; j<5; j++)
   if (rawscore >= rawScores[j]) break;
  }
  grade = grades[j];
  cout << idNum << " " << grade << endl;
 }
```

Ansr: need to do rounding up..

```
int main () {
 int rawScores[5]....
 char grade[5]....
 double rawscore;
 long idNum;
 char grade;
 while (cin >> idNum >> rawscore) {
  for (int j=0; j<5; j++)
   if ((rawscore+.5) >= rawScores[j]) break;
  }
  grade = grades[j];
  cout << idNum << " " << grade << endl;
 }
```

Chapter 10 — Design Exercises

1. Reverse Order

Acme Corporation has a **master.txt** file of item numbers and their corresponding costs. The file has been sorted into ascending order on the item numbers. However, in order to compare their items with a competitor, the printout must be in high to low order based on the item numbers. Management does not want to spend the computer time to actually resort the file because it is a very large one. Write a program that inputs the **long** item numbers and **float** costs into two arrays that can handle a maximum of 10,000 items. Then print the report listing the item number and associated cost.

A solution:

```
long itemNum[10000];
float cost[10000];
open infile using master.txt
set j = 0
while (j<10000 && infile >> itemNum[j] >> cost[j]) j++;
for (int k=j-1; k>=0; k--) {
     cout << itemNum[k] << cost[k];
}
close infile
```

2. Statistical Study of a Car's Mpg (miles per gallon)

For 36 months, a record has been kept of gas station fill-ups. Specifically, the driver logged the date (mm-dd-yyyy), the odometer reading in miles and the number of gallons of gas it took to fill the car. It is postulated that one can use the average monthly mile per gallons (mpg) figures to determine when the car needs maintenance. That is, when the mpg decreases substantially, it is indicative of at least a tuneup being required. Write a program to prove or disprove the theory.

Input the **log.txt** file whose lines consist of the date, the mileage and the number of gallons. For each line, calculate the mpg. Accumulate the mpg figures until the month changes and then find that month's average mpg. Store that month's average mpg into an array which can hold up to 36 values. The very first line of the file is special. It represents the very first full tank of gas, so no average can be calculated for this entry.

Now print a report listing each month's average mpg and print a ** beside that mpg which is more than 20% different from the previous month. The report might appear like this.

```
Avg Monthly Mpg
     35.6
```

```
36.5
25.5 **
34.5 **
33.9
```

A Solution:

main defines:

infile, miles, prevMiles, gallons, mpg, monthlyMpg, month, day, year, prevMonth, numFills, numMonths, avgMpg[36], sum;

```
open infile using file log.txt
input from infile month, day, year, miles, gallons
prevMonth = month;
prevMiles = miles;
numFills = 0;
numMonths = 0;
while (input from infile month, day, year, miles, gallons is successful) do
 if (month != prevMonth) {
  avgMpg[numMonths] = sum / numFills;
  numMonths++;
  prevMonth = month;
  sum = 0;
  numFills = 0;
 }
 mpg = (miles - prevMiles) / gallons;
 sum += mpg;
 prevMiles = miles;
 numFills++;
end while
close infile
display heading line
display avgMpg[0];
for (int k=1; k<numMonths; k++) {
        display avgMpg[k]
        if (avgMpg[k] < avgMpg[k-1] * .8 ||
           avgMpg[k] > avgMpg[k-1] * 1.2)
                Display a *
}
```

Chapter 10—Stop! Do These Exercises Before Programming

1. Acme National Sales expanded and their parallel arrays of state codes and corresponding tax rates have grown larger. The programmer has coded the following to load the tables. Why does it not work? What must be done to fix the coding so that the tables are correctly loaded?

```
const int MaxLimit = 50;
int LoadArrays (int stateCode[], taxRate[], int limit);
int main () {
 double taxRate[50];
 int stateCode[50];
 int numValues;
 LoadArrays (stateCode, taxRate, 50);
 ...
int LoadArrays (int stateCode[], taxRate[], int limit) {
 ...// gets infile opened
 int j=0;
 while (infile >> stateCode[limit] >> taxRate[limit])
  j++;
 infile.close();
 return limit;
}
```

Ansr: need to use a variable subscript and pass MaxLimit not 50 and guard against too many elements; return j

```
const int MaxLimit = 50;
int LoadArrays (int stateCode[], taxRate[], int limit);
int main () {
 double taxRate[50];
 int stateCode[50];
 int numValues;
 LoadArrays (stateCode, taxRate, MaxLimit);
 ...
int LoadArrays (int stateCode[], taxRate[], int limit) {
 ...// gets infile opened
 int j=0;
 while (j<limit && infile >> stateCode[j] >> taxRate[j])
  j++;
 infile.close();
 return j;
}
```

2. With the arrays loaded, the programmer embarked on the writing of a sort function to get the state codes in numerical order for faster table look ups within the main program. The following function is to sort the state codes into increasing numerical order. It does not work properly. Why? How can it be fixed up?

```
void SortStateCodes (int state[], int limit) {
 int j, k, temp;
 for (j=0; j<limit; j++) {
  for (k=j; k<limit; k++) {
   state[j] = temp;
   temp = state[k];
   state[k] = state[j];
  }
 }
}
```

Ansr: need to have j go to limit-1 and k goes from j+1 on. The swap is in incorrect order

```
void SortStateCodes (int state[], int limit) {
 int j, k, temp;
 for (j=0; j<limit-1; j++) {
  for (k=j+1; k<limit; k++) {
   temp = state[k];
   state[k] = state[j];
   state[j] = temp;
  }
 }
}
```

3. Now that the states are sorted successfully, the programmer realized that the tax rates needed to be sorted as well. Tired at the end of a long day, he produced the following patch to his program. Assume that the **SortTaxRates()** function now works perfectly.

```
const int MaxLimit = 50;
int LoadArrays (int stateCode[], taxRate[], int limit);
void SortStateCodes (int states[], int limit);
void SortTaxRates (double rates[], int limit);
int main () {
 double taxRate[MaxLimit];
 int stateCode[MaxLimit];
 ...
 SortStateCodes (stateCode, numValues);
 SortTaxRates (taxrate, numValues);
 ...
```

When he tested the program, all of the taxes were completely wrong. What is wrong with the design of the entire sorting method? How can it be fixed? Write the sorting code that would enable the program, given a state code, find the correct tax rate. (However, do not write any table look up coding.)

Ansr: the two arrays are parallel arrays and must be treated as such in the sort.

```
void SortStateCodes (int state[], double rates[], int limit)
{
 int j, k, temp;
```

```
 double trate;
 for (j=0; j<limit-1; j++) {
  for (k=j+1; k<limit; k++) {
   temp = state[k];
   state[k] = state[j];
   state[j] = temp;
   trate = rates[k];
   rates[k] = rates[j];
   rates[j] = trate;
  }
 }
}
```

4. Grateful for your assistance in bailing him out of his coding mess yesterday, the programmer today has embarked upon construction of a fast matching function that is given the two arrays and the number of elements in it and returns the tax rate. This is what has been produced thus far.

```
double FindTaxRate (int states[], double rates[], int num,
                    int findState) {
 int j = 0;
 double taxRate;
 while (j<num) {
  if (findState == states[num]) {
   taxRate = rates[j];
  }
  j++;
 }
 return taxRate;
}
```

The function has serious design flaws and of course does not work properly. Why? What must be done to correct this coding so that it produces the correct taxes in all circumstances?

Ansr: wrong subscript in loop and not ending the loop when the match is found

```
double FindTaxRate (int states[], double rates[], int num,
                    int findState) {
 int j = 0;
 while (j<num) {
  if (findState == states[j]) return rates[j];
  j++;
 }
 return 0; // nothing found
}
```

5. Looking over Practice Problem 4 just above, the **FindTaxRate()** function does not in any way

make use of the fact that the state codes are sorted into numerical order. Using a binary search function would provide the speed up desired. Assume that the **BinarySearch()** function presented earlier in this chapter has been rewritten using an array of **int**s instead of **long**s. Rewrite the **FindTaxRate()** function above to use the **BinarySearch()** in finding the tax rate to return.

Ansr:

```
double FindTaxRate (int states[], double rates[], int num,
                    int findState) {
 int j;
 if (BinarySearch (states, num, findState, j))
  return rates[j];
 return 0; // nothing found
}
```

Note the BinarySearch is expecting the array to be long not int. So one would have to slightly modify the routine as given in this chapter.

Chapter 11 — Design Exercises

1. Design a Grade Book Program

The Grade Book Program inputs a set of students grades for a semester. First, design the layout of the data file to be used for input and then design the program to produce the Grade Report shown below.

The data consists of a student id number which is their social security number, their name which can be up to 20 characters long, the course name which can be up to 10 characters in length, the course number and finally the letter grade earned. Design how the input lines must be entered. Include in what order they are entered; pay particular attention to specifically how the student names are going to be entered on your lines.

The Grade Report produced by the program that is to input your data file appears as follows.

```
     Student Grade Report

 Student    Student              ----Course-----
    Id      Name                 Name     Number Grade
 111111111  Sam J. Jones         Cmpsc      125    A
...
```

A solution:
input line: 999999999 “john jones” cmpsc 125 a
main:
variables

```
const int NAMELEN = 21;
const int CRSELEN = 11;
long id;
char name[NAMELEN];
char course[CRSELEN];
short courseNum;
char grade;
char c; // for the “
open infile
display heading and two col heading lines
while (infile >> id >> c) {
  infile.getline (name, sizeof(name), ‘\”’);
  infile >> course >> courseNum >> grade;
  grade = toupper (grade);
  display id
  set left justification
```

```
  display  name, course
  set right justification
  display courseNum, grade
}
close infile
```

2. Design the Merge Conference Roster Program

Two sections of a conference course have been merged into one larger section. Each original section has a file of the attendee names. You are to write a program that merges the two into one new file. Each original file contains, in alphabetical order, the attendee names which can be up to 30 characters long, one name per line. The new file this program creates must also be in alphabetical order.

A solution:

```
main variables infile1, infile2, outfile
        const int MAX = 31;
        char name1[MAX], name2[MAX];

    open infile1, infile2, and outfile
    infile1.getline (name1, sizeof(name1));
    infile2.getline (name2, sizeof(name2));
    while (infile1 && infile2) {
        if (stricmp (name1, name2) < 0) {
            outfile << name1 << endl;
            infile1.getline (name1, sizeof(name1));
        }
        else {
            outfile << name2 << endl;
            infile2.getline (name2, sizeof(name2));
        }
    }
    while (infile1)  {
        outfile << name1 << endl;
        infile1.getline (name1, sizeof(name1));
    }
    while (infile2) {
        outfile << name2 << endl;
        infile2.getline (name2, sizeof(name2));
    }
    close infile1, infile2, outfile
```

Chapter 11 — Stop! Do These Exercises Before Programming

1. A programmer needs to input the day of the week as a character string. The following coding failed to run properly. Why? What must be done to fix it up?

```
char dayName[9];
cin >> dayName;
```

Ansr: Wednesday is 9 chars, null terminator wipes out byte 10!

```
char dayName[10];
```

2. A program needs to input the chemical compound names of two substances and then compare to see if the names are the same. The following was coded and compiles without errors but when run always produces the wrong results. Why? How can it be fixed?

```
char compound1[40];
char compound2[40];
infile1.get (compound1, sizeof (compound1));
infile2.get (compound2, sizeof (compound2));
if (compound1 == compound2) {
  cout << "These compounds match\n";
else
  cout << "These compounds do not match\n";
```

Ansr: names of arrays are mememory addresses of first elements. Use string functions

```
if (stricmp (compound1, compound2) == 0) {
  cout << "These compounds match\n";
else
  cout << "compounds do not match\n";
```

3. The programmer inputted a compound name and its cost and then wanted to check to see if it was equal to “Sodium Chloride.” The following coding compiles with no errors but when it runs, it fails to find Sodium Chloride when that is input. The input line is

```
Sodium Chloride      4.99
```

What is wrong and how can it be fixed?

```
char compound[20];
double cost;
cin.get (compound, sizeof (compound));
cin >> cost;
if (stricmp (compound, "Sodium Chloride") == 0) {
 cout << "Found\n"
}
```

Ansr: the input compound has 5 blanks on the end of it and thus is not equal.

```
cin.get (compound, sizeof (compound));
int j = strlen (compound) -1;
while (j>=0 && compound[j] == ' ') j--;
```

```
compound[j+1] = 0;
cin >> cost;
```

4. The input file consists of a **long** student id number followed by a blank and then the student's name. The following coding does not input the data properly. Why? What specifically is input when the user enters a line like this?

```
1234567 Sam Spade<cr>
```

How can it be fixed so that it correctly inputs the data?

```
long id;
char name[20];
while (cin >> id) {
  cin.get (name, sizeof (name));
 ...
}
```

Ansr: the input stream is pointing to the blank after the id when the get is called. Need to skip over that blank

```
while (cin >> id >> ws) {
```

5. A file of student names and their grades is to be input. The programmer wrote a **GetNextStudent()** function. It does not work. How can it be fixed so that it does work properly?

```
char name[20];
char grade;
while (GetNextStudent (infile, name, grade, 20)) {
...
istream GetNextStudent (istream infile, char name[],
                        char grade, int maxLen) {
 infile.get (name, sizeof (name));
 infile.get (grade);
 return infile;
}
```

Ansr: sizeof (ptr) returns 4 not 20; also need to skip over the blank between name and grade

```
 infile.get (name, maxLen);
 infile >> grade;
```

6. The proposed Acme Data Records consist of the following.

```
12345 Pots and Pans 42 10.99
23455 Coffee #10 can 18 5.99
32453 Peanuts 20 1.25
```

The first entry is the item number, the second is the product description, the third is the quantity on hand, and the fourth is the unit cost. Assume that no description can exceed 20 characters. The programmer wrote the following code to input the data.

```
int main () {
 long   id;
 char   description[21];
```

```
      int    quantity;
      double cost;
      ifstream infile ("master.txt", ios::in | ios::nocreate);
      while (infile >> id >> description >> quantity >> cost) {
     ...
```

However, it did not run at all right. What is wrong with it? Is it possible to fix the program so that it would read in that data file? What would you recommend?

Ansr: the blanks in the description field end the string extraction. There is no way to tell where each description string ends as it is given in the file. One way around it is to insert “ ” around the description field in the input file.

Chapter 12 — Design Exercises

1. Spreadsheet Design

You want to track your budget for a year's time on a monthly basis. You have categorized your expenses into ten categories and have only one job producing income. The input file consists of 11 lines. The first is the income line. It contains a character string description, "Monthly Income," which is then followed by 12 monthly income figures. The next 10 lines contain the expense description, such as "House Rent" followed by the 12 monthly figures for that expense. Sketch out the Top-Down Design and pseudo coding to produce your budget report shown below.

```
          My Budget Report

Item                Jan     Feb     Mar   ... Dec
Monthly Income   999.99  999.99  999.99     999.99
House Rent       999.99  999.99  999.99     999.99
...
                 ------  ------ ------     ------
Total Expenses  9999.99 9999.99 9999.99   9999.99
Net Profit      9999.99 9999.99 9999.99   9999.99
```

You should make effective use of functions to eliminate as many repetitive actions as possible.

Design: main (produce the report), LoadBudget, DoCalculations, PrintBudget
main storage for main:

```
const int MAXROWS = 13;
const int MAXCOLS = 12;
const int ITEMLEN = 31;
const int TotExpIdx = 11;
const int NetProfIdx = 12;
double budget[MAXROWS][MAXCOLS];
char item[MAXROWS][ITEMLEN];
strcpy (item[TotExpIdx], "Total Expenses");
strcpy (item[NetProfIdx], "Net Profit");
LoadBudget (item, budget);
DoCalculations (budget);
PrintBudget(item, budget);
```

```
void LoadBudget (char item[][ITEMLEN], double budget[][MAXCOLS]) {
    open infile
    for (int j=0; j<11; j++) {
        infile >> ws;
        infile.get(item[j], ITEMLEN);
```

```
                for (int k=0; k<MAXCOLS; k++) {
                        infile >> budget[j][k];
                }
        }
}

void DoCalculations (double budget[][MAXCOLS]) {
        const int TotExpIdx = 11;
        const int NetProfIdx = 12;
        sum = 0;
        for (int col=0; col<12; col++) {
                sum = 0;
                for (int row=1; row<11; row++)
                        sum += budget[row][col];
                budget[TotExpIdx][col] = sum;
                budget[NetProfIdx][col] = budget[0][col] - sum;
        }
}

void PrintBudget (char item[][ITEMLEN], double budget[][MAXCOLS]) {
        display heading and column heading lines
        for (int row=0; row<MAXROWS-2; row++) {
                display item[row];
                for (int col=0; col<MAXCOLS; col++) {
                        display budget[row][col];
                }
                display a newline
        }
        display a dash line
        for (int row=TotExpIdx; row<MAXROWS; row++) {
                display item[row];
                for (int col=0; col<MAXCOLS; col++) {
                        display budget[row][col];
                }
                display a newline
        }
}
```

Chapter 12—Stop! Do These Exercises Before Programming

1. The programmer wanted to define a two-dimensional array of grades. There are five sections of twenty-four students each. What must be done to the following to get it defined correctly?

```
const int Sections = 5;
const int Students = 24;
int main () {
 char grades[Students, Sections];
```

Which array bounds should come first, assuming that the normal processing handles all of the students within a given section at one time? Why?

Ansr: `char grades[Sections][Students];`

2. Since not every section has 24 students in it, the programmer decided to have another array called **numberStudentsInThisSection** which is an array of five integers, one for each section. Thus, **numberStudentsInThisSection[0]** contains the number of students in that section. With this defined, a **LoadStudentArrays()** function was written but does not compile or work. Why? What must be done to make this work properly?

```
const int Sections = 5;
const int Students = 24;
int LoadStudentArrays (char grades[][Students],
                       numberStudentsInThisSection[],
                       int maxSections, int maxStudents);
int main () {
 char grades[Sections, Students];
 int numberStudentsInThisSection[Sections];
 int numSections = LoadStudentArrays (grades[][Students],
         numberStudentsInThisSection[], Sections, Students);
...
int LoadStudentArrays (char grades[][Students],
                       numberStudentsInThisSection[],
                       int maxSections, int maxStudents){
 int j = 0;  // section subscript
 while (cin >> ws) {
  int k = 0; // student subscript
  while (cin >> grades[k][j]) {
   k++;
  }
  numberStudentsInThisSection[j] = j;
  j++;
 }
 return j;
}
```

Ansr: confusion on invoking the functions versus prototypes; mixed up subscripts

```
const int Sections = 5;
```

```
const int Students = 24;
int LoadStudentArrays (char grades[][Students],
                       numberStudentsInThisSection[],
                       int maxSections, int maxStudents);
int main () {
 char grades[Sections][Students];
 int numberStudentsInThisSection[Sections];
 int numSections = LoadStudentArrays (grades,
        numberStudentsInThisSection, Sections, Students);
...
int LoadStudentArrays (char grades[][Students],
                       numberStudentsInThisSection[],
                       int maxSections, int maxStudents){
 int j = 0;  // section subscript
 while (cin >> ws) {
  int k = 0; // student subscript
  while (cin >> grades[j][k]) {
   k++;
  }
  numberStudentsInThisSection[j] = k;
  j++;
 }
 return j;
}
```

3. Next the **main()** function attempted to printout all of the grades to see if they had been input properly. The following coding does not work properly. Why? What must be done to get it to properly print out the grades as entered?

```
const int Sections = 5;
const int Students = 24;
int LoadStudentArrays (char grades[][Students],
                       numberStudentsInThisSection[],
                       int maxSections, int maxStudents);
int main () {
 char grades[Sections, Students];
 int numberStudentsInThisSection[Sections];
 int numSections = LoadStudentArrays (grades[][Students],
        numberStudentsInThisSection[], Sections, Students);
 for (int j=0; j<numSections; j++) {
  cout << "\n\nSection: " << j << endl;
  for (int k=0; k<numberStudentsInThisSection[k]; k++) {
   cout << grades[k][j] << endl;
  }
 }
```

Ansr: wrong subscript in the for loop and reversed subscripts in the cout; probably section number should start at 1

```
int main () {
 char grades[Sections][Students];
 int numberStudentsInThisSection[Sections];
 int numSections = LoadStudentArrays (grades,
        numberStudentsInThisSection, Sections, Students);
 for (int j=0; j<numSections; j++) {
  cout << "\n\nSection: " << j+1 << endl;
  for (int k=0; k<numberStudentsInThisSection[j]; k++) {
   cout << grades[j][k] << endl;
  }
 }
```

4. With the data properly input and printed, the next step is to calculate the average grade for each section. Since the grades are letter grades, assume that a 4.0 system is in use. That is, an A is worth 4 points, B is 3 and so on. The **FindAvgGrades()** function does not compile. Why? How can it be made to work properly?

```
const int Sections = 5;
const int Students = 24;
void FindAvgGrades (char grades[][Students],
                    numberStudentsInThisSection[],
                    int numSections, double avgs[]);
int main () {
 char grades[Sections, Students];
 int numberStudentsInThisSection[Sections];
 int numSections;
 double averages;
 ...
 FindAvgGrades (grades, numberStudentsInThisSection[],
                int numSections, averages);
 ...
void FindAvgGrades (char grades[][Students],
                    int numberStudentsInThisSection[],
                    int numSections, double avgs[]){
 double sum;
 for (j=0; j<numberStudentsInThisSection[j]; j++) {
  sum = 0;
  for (k=0; k<numberStudentsInThisSection[j]; k++) {
   switch (grades[j[k]) {
    case 'A':
     sum += 4;
    case 'B':
     sum += 3;
    case 'C':
     sum += 2;
    case 'D':
     sum += 1;
```

```
        case 'F':
         sum += 0;
       }
      }
      avgs[k] = sum / numberStudentsInThisSection[j];
     }
    }
```

Ansr: call to FindAvgGrades confuses prototype with array; main — averages not an array; j loop should be numSections; syntax on switch expression; no breaks in case; averages subscript should be j

```
    int main () {
     char grades[Sections][Students];
     int numberStudentsInThisSection[Sections];
     int numSections;
     double averages[Sections];
     ...
     FindAvgGrades (grades, numberStudentsInThisSection,
                    numSections, averages);
     ...
    void FindAvgGrades (char grades[][Students],
                        int numberStudentsInThisSection[],
                        int numSections, double avgs[]){
     double sum;
     for (j=0; j<numSections; j++) {
      sum = 0;
      for (k=0; k<numberStudentsInThisSection[j]; k++) {
       switch (grades[j][k]) {
        case 'A':
         sum += 4; break;
        case 'B':
         sum += 3; break;
        case 'C':
         sum += 2; break;
        case 'D':
         sum += 1; break;
        case 'F':
         sum += 0; break;
       }
      }
      avgs[j] = sum / numberStudentsInThisSection[j];
     }
    }
```

5. Sorting of a two-dimensional array usually means sorting each row's worth of column values into order. Assume that there are 10 rows of raw scores and each row has 20 columns and that all

elements are present. That is, there are 200 values in the array. The following coding to sort the array fails. Why? How can it be fixed so that the data are sorted properly?

```
const int Rows = 10;
const int Cols = 20;
double rawScores[Rows][Cols];
for (int j=0; j<Rows; j++) {
 double temp;
 for (int k=0; k<Cols; k++) {
  for (int m=k; m<Cols; m++) {
   if (rawScores[j][k] < rawScores[j][m]) {
    temp = rawScores[k][j];
    rawScores[j][k] = rawScores[j][m];
    rawScores[j][m] = temp;
  }
 }
}
```

Ansr: the subscripts ranges are off on the k-m for stmts; test for switching reversed;

```
for (int j=0; j<Rows; j++) {
 double temp;
 for (int k=0; k<Cols-1; k++) {
  for (int m=k+1; m<Cols; m++) {
   if (rawScores[j][m] < rawScores[j][k]) {
    temp = rawScores[j][k];
    rawScores[j][k] = rawScores[j][m];
    rawScores[j][m] = temp;
  }
 }
}
```

Chapter 13 — Design Exercises

1. Airline Scheduling Program

Acme Airline wants a new program to track their arrival and departure schedules. Create a **DATE** structure to contain a date that consists of three numbers: month, day and year. Create a **TIME** structure to contain a time that consists of two numbers: hours and minutes. All times are on a 24-hour basis; that is, 10:00 p.m. would be entered as 22:00. Next, create a **FLIGHT** structure that contains the departure date and time, the arrival date and time, the character string flight number, the maximum number of passengers and the current number of passengers. The arrival and departure dates and times should be instances of your **DATE** and **TIME** structures. The flight number is a string of 10 characters maximum including the null-terminator.

Now write the sketch for the program that inputs a set of data on flights, stores them in an array of **FLIGHT** structures. Then, print a report of the flights as entered. Each line of the input file consists of the flight number, departure date and time, arrival date and time, maximum passengers and current number of passengers.

A solution:

```
struct DATE {
  int month;
  int day;
  int year;
};
struct TIME {
  int hours;
  int minutes;
};
const int FLIGHTLEN = 10;
struct FLIGHT {
  DATE departdate;
  TIME departtime;
  DATE arrivedate;
  TIME arrivetime;
  char flightName[FLIGHTLEN];
  int maxPassengers;
  int numPassengers;
};
```

main

```
const int MAX = 100;
FLIGHT flight[MAX];
int num = LoadFlights (flight, MAX);
```

```
        PrintReport (flight, num);

int LoadFlights (FLIGHT flight[], int limit) {
        int j = 0;
        open infile
        while (j<limit && infile >> ws) {
                infile.get(flight[j].flightName, FLIGHTLEN);
                infile >> flight[j].departdate.month >>flight[j].departdate.day
                >> flight[j].departdate.year >> flight[j].departtime.hours
                >>flight[j].departtime.minutes>> flight[j].arrivedate.month
                >> flight[j].arrivedate.day >> flight[j].arrivedate.year
                >> flight[j].arrivetime.hours >> flight[j].arrivetime.minutes
                >> flight[j].maxPassengers>> numPassengers;
                j++;
        }
        close infile
        return j;
}

void PrintReport (FLIGHT flight[], int num) {
        display heading and column headings
        for (j=0; j<num; j++) {
          display flight[j].flightName, flight[j].departdate.month, flight[j].departdate.day,
          flight[j].departdate.year, flight[j].departtime.hours,
          flight[j].departtime.minutes, flight[j].arrivedate.month, flight[j].arrivedate.day,
          flight[j].arrivedate.year, flight[j].arrivetime.hours, flight[j].arrivetime.minutes,
          flight[j].maxPassengers, numPassengers
        }
}
```

2. Sports Event Reporting

For the time trails at a major sporting event, a series of programs needs to be designed and made to track the results of the contestants. The judges desire to store the contestant's three digit id number, the contestant name of up to 30 characters, and the start and stop times. A time is input as hh:mm:ss.ssss. First design a **TIME** structure to store the three portions of a time. Then, design an **EVENTREC** structure to store the basic information for one contestant. There is one additional field in the **EVENTREC**, the total elapsed time in seconds.

Now sketch the Data Entry program that inputs **EVENTREC** data and writes a binary master file of the contestant data. It must calculate the total elapsed time in seconds.

Now sketch an Event Summary Program that inputs the binary master file into an array of **EVENTREC** structures. The array should then be sorted into increasing total elapsed time order. The printed report displays the contestant id number, name and total elapsed time.

A solution:

```
struct TIME {
   short hrs;
   short min;
   float sec;
};
const int NAMELEN = 31;
struct EVENTREC {
  short id;
  char name[NAMELEN];
  TIME start;
  TIME stop;
  double elapsedTime;
};
```

main program Data Entry

```
EVENTREC rec;
char c;
open infile, outfile
while (infile >> rec.id >> ws) {
        infile.get(rec.name, sizeof (rec.name));
        infile >> rec.start.hrs >> c >> rec.start.min >> c >> rec.start.sec
              >> rec.stop.hrs >> c >> rec.stop.min >> c >> res.start.sec;
        rec.elapsedTime = (rec.stop.hrs*3600 + rec.stop.min*60 + rec.stop.sec) -
                          (rec.start.hrs*3600 + rec.start.min*60 + rec.start.sec);
        outfile.write ((char*) &rec, sizeof (rec));
}
```

main program Event Summary

```
const int MAX = 1000;
EVENTREC rec[MAX];
open infile
int num = 0;
while (infile.read ((char*) &rec[num], sizeof (EVENTREC))) {
 num++;
}
close infile
SortRecs (rec, num);
PrintRecs (rec, num);
```

```
void SortRecs (EVENTREC rec[], int num) {
        for (j=0; j<num-1; j++) {
                for (k=j+1; k<num; k++) {
                        if (rec[k].elapsedTime < rec[j].elapsedTime) {
                                EVENTREC t = rec[k];
                                rec[k] = rec[j];
                                rec[j] = t;
                        }
                }
        }
}

void PrintRecs (EVENTREC rec[], int num) {
        display heading and column headings
        for (j=0; j<num; j++) {
                display rec[j].id, rec[j].name, rec[j].elapsedTime
        }
}
```

Chapter 13—Stop! Do These Exercises Before Programming

1. Acme wishes to make a database of all of its client's TV preferences. A TV preference consists of the **short int** channel number, two **short int**s for the hour and minute the show is broadcast but are stored in 24-hour format (that is, 8pm is stored as 20 hours), a **short int** for the day of the week, and the name of the show which can hold thirty characters. The programmer coded the following which does not compile among other goofs. Correct this sequence so that it compiles and meets the specifications above.

```
Structure PREFERENCE {
     short channel
     short day
     short hr
     short min
     char name[MAXLEN]
}
const int MAXLEN = 30;
```

Ansr: const int must occur before struct that uses it; length is 31; watch case; ends in ; and data menber definitions end in ;

```
const int MAXLEN = 30;
structure PREFERENCE {
     short channel;
     short day;
     short hr;
     short min;
     char  name[MAXLEN]
};
```

2. Just to test the structure coding, the programmer decided to see if he could input one set of preferences and then display it. However, his attempt met with complete failure and does not compile. Why? What would have to be done to make this work?

```
int main () {
  cin >> PREFERENCE.channel >>  PREFERENCE.day
      >> PREFERENCE.hr      >> PREFERENCE.min
      >> PREFERENCE.name;
  cout << PREFERENCE.channel << PREFERENCE.day
       << PREFERENCE.hr      << PREFERENCE.min
       << PREFERENCE.name << endl;
```

Ansr: we need an instance of the stucture, not its model or template

```
int main () {
  PREFERENCE p;
  cin >> p.channel >> p.day >> p.hr >> p.min >> p.name;
  cout << p.channel << p.day << p.hr << p.min << p.name
       << endl;
```

3. The problem specifications called for creating a **LoadPreferences()** function. Here the programmer ran into a great deal of difficulty. What is wrong with this function? How can it be fixed?

```
void LoadPreferences (PREFERENCE rec, int limit,
                      istream& infile) {
 for (int j=0; j<limit && infile >> ws; j++) {
  infile >> rec[j].channel >> rec[j].day >> rec[j].hr
         >> rec[j].min >>rec[j].name;
 }
}
```

Here are two typical data lines.

```
11 6 08:00pm Babylon 5
12 5 07:30pm Doctor Who
```

Ansr: recs is not an array; need to input the : and pm and name has blanks in it.

```
void LoadPreferences (PREFERENCE rec[], int limit,
                      istream& infile) {
 char c;
 for (int j=0; j<limit && infile >> ws; j++) {
  infile >> rec[j].channel >> rec[j].day >> rec[j].hr
         >> c >> rec[j].min >> c >> c >> ws;
  infile.get (rec[j].name, sizeof (rec[j].name));
 }
}
```

4. Having gotten the array loaded, the programmer decided to print what was in the array to be sure the data had been entered correctly. However, it does not compile. How can it be made to work properly?

```
int main () {
 PREFERENCE recs[1000];
 int numRecs;
 for (int j=0; j<numRecs; j++) {
  cout << recs.channel << setw (5) << recs.day << " "
       << setfill ('0') << setw (2) << recs.hr << ':'
       << recs.min << setfill (' ') << setw (40)
       << recs.name << endl;
 }
```

Ansr: forgot the subscripts

5. Next, management wanted the entire array sorted into order based on the TV show's name so that the program could then calculate frequencies of each TV show. The **sortArray()** function does not work properly, though it compiles. What is wrong and how can it be fixed?

```
void sortArray (PREFERENCE recs[], int numRecs) {
 PREFERENCES temp;
```

```
 for (int j=0; j<numRecs; j++) {
  for (int k=j; k<numRecs; k++) {
   if (recs[k].name < recs[j].name) {
    temp = recs[k];
    recs[k] = recs[j];
    recs[j] = temp;
   }
  }
 }
}
```

Ansr: must use **stricmp** to compare strings; j loop doing one too many rows and k loop starting too soon.

```
void sortArray (PREFERENCE recs[], int numRecs) {
 PREFERENCES temp;
 for (int j=0; j<numRecs-1; j++) {
  for (int k=j+1; k<numRecs; k++) {
   if (stricmp(recs[k].name, recs[j].name) < 0) {
    temp = recs[k];
    recs[k] = recs[j];
    recs[j] = temp;
   }
  }
 }
}
```